U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

MSc

MSc
2.º CICLO
FCUP
2017

U.PORTO

CompAlg – Ferramenta de Ensino e Aprendizagem da Lógica de Programação

Augusto Manuel Bilabila

FC

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

CompAlg – Ferramenta de Ensino e Aprendizagem da Lógica de Programação

Augusto Manuel Bilabila,

Dissertação de Mestrado apresentada à
Faculdade de Ciências da Universidade do Porto em
Ciência de Computadores

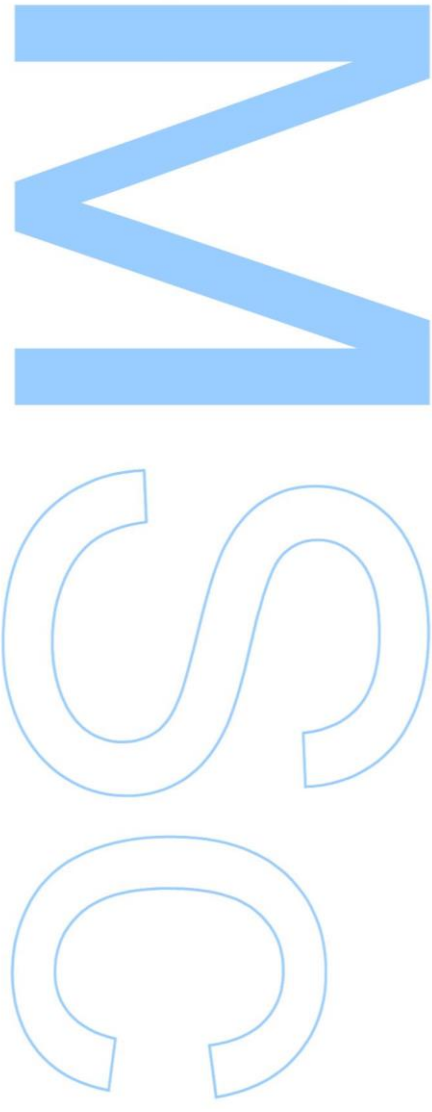
2017

MSc

CompAlg – Ferramenta de Ensino e Aprendizagem da Lógica de Programação

Augusto Manuel Bilabila
Mestrado em Ciência de Computadores
Departamento de Ciência de Computadores
2017

Orientador
Pedro Ribeiro, professor auxiliar, FCUP



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Para a minha família, meus irmãos na fé, amigos e colegas de profissão.

Agradecimentos

Em primeiro lugar gostaria de expressar minha gratidão ao meu orientador, o professor Pedro Manuel Pinto Ribeiro por ter estado comigo ao longo de todo esse trabalho, por ter acreditado em mim e nunca ter desistido de mim e dos meus sonhos. Pela coragem e amizade que prestou em todos os momentos. O meu profundo agradecimento.

Em segundo lugar ao meu amor, Rízia Rodrigues da Silva, que num momento tão crítico e instável da minha vida esteve sempre comigo me suportando e amando nos momentos amargos dessa caminhada.

Em terceiro lugar as famílias Popusoi e Pego, que mesmo sendo estrangeiro me acolheram e cuidaram de mim como um filho, como um verdadeiro membro da família, não há preço que pague todo amor que ganhei.

Finalmente, queria agradecer à todos os membros da Igreja Evangélica Batista de Cedofeita, pelo carinho, pelo amor e pela amizade. A todos que direta e indiretamente ajudaram-me nessa luta, a todos que comigo choraram e sorriram. Aos jovens dessa Igreja, o meu muito obrigado. A minha família, amigos e colegas que tanto me ajudaram, em especial o Ricardo Leite, Anabel, Ahmad, Artur Paniche, Marcos Pais, Brian, Revailton Júnior, Ricardo Alves, Fábio Carvalho, etc. O meu muitíssimo obrigado.

Abstract

Nowadays, learning **computer programming** is vital in all areas. The technological and industrial evolution demands well prepared professionals, and without a doubt **programming knowledge** is a crucial factor in obtaining productivity and being efficient.

The process of teaching programming logic has been a tough task, like it is described by some authors. This has been giving origin to a large amount of research about the best way to teach and learn how to program. Many researchers defend the notion of **improving the teaching methodology through repetition of well defined stages**, while others support **the inclusion of tools with the purpose of helping programming education**.

The problem of learning how to program is even more complicated on the **African Countries with Portuguese as Official Language (PALOP)**. This is because, even though the teaching methodologies work relatively well, the supporting tools do not, because the vast majority of them **are developed with english keywords**, making them harder and less suited for **native portuguese language** students.

This thesis presents the development of a tool aimed towards helping in programming education, combining a methodology that is pedagogically accepted into an intuitive and easy to use tool for an effective teaching and learning process, with a programming language fully written in portuguese, based on Portugol (a pseudo-language with keywords in portuguese). This also allows a self learning programming experience, where a student can try to learn by itself.

Although other tools have also been developed (in Brazil and Portugal) with the same goal, they were developed with other realities of people and teaching environments. In this thesis we focus specifically on the PALOPs case, and in particular on the case of Angola. We aim to help to teach how to program not only at the higher education level, but also on all areas that have programming as part of their curriculum.

In short, we want to present **an integrated development environment for teach-**

ing and learning, **not as a productivity tool**, but has **a totally in portuguese tool exclusively aimed towards learning and/or teaching programming logic**, allowing to write programs using portuguese keywords, to execute a program step by step, to visualize the content of variables, and to profile a program. Besides this, the tool supports the conversion of source code, written in our language (in portuguese), to the C and Java languages, allowing the CompAlg student to have an early on contact with two of the most used languages in Angola's universities. The developed tool has also a complete support manual, describing the language and the graphical interface, as well as online help through email.

Resumo

A aprendizagem da **programação de computadores** torna-se vital nos dias de hoje em todas as áreas. A evolução tecnológica e industrial exige cada vez mais profissionais melhores preparados para suportar a demanda, e sem dúvida que o **conhecimento da programação** é um fator crucial para maior rendimento e diligência no que se pretende desenvolver.

O processo de ensino da lógica de programação de computadores tem sido de difícil execução, tal como escrevem alguns atores, envolvidos no processo. Tal fato tem levantado um conjunto de pesquisas sobre a melhor forma de ensinar e aprender a programação, muitos investigadores defendem que **a melhoria da metodologia de ensino através de repetições de etapas bem definidas** poderia ajudar no aprendizado, mas por outro lado existe um grupo de investigadores que defendem **a inclusão de ferramentas com propósito de auxiliar no ensino da programação**.

O problema da aprendizagem da programação torna-se ainda mais complicado nos **Países Africanos de Língua Oficial Portuguesa (PALOP)** isso porque, embora algumas metodologias de ensino funcionem razoavelmente bem, as ferramentas de auxílio não, porque na sua grande maioria **são desenvolvidas com palavras chave em inglês** dificultando o aprendiz de **língua nativa portuguesa**.

Esta tese apresenta o desenvolvimento de uma ferramenta de auxílio a programação, que faz a combinação de uma metodologia pedagogicamente aceite numa ferramenta de uso fácil e intuitivo para um ensino e aprendizagem eficaz, com uma nova linguagem de programação baseada no Portugol (Pseudo-linguagem com palavras chave em português), totalmente em português. E ainda permitindo que o aprendiz autodidata possa aprender a lógica de programação por si mesmo.

Embora várias outras ferramentas tenham sido desenvolvidas (no Brasil e em Portugal) com a mesma finalidade, foram desenhadas e desenvolvidas para atender uma determinada realidade de pessoas e de ensino. Nós apresentamos nessa tese uma realidade que atende os PALOPs, principalmente Angola. Tanto para o ensino superior como também para o ensino médio, em todas as áreas que têm a programação no seu plano

curricular.

Em suma, queremos apresentar **um ambiente de desenvolvimento integrado de ensino-aprendizagem**, **não como uma ferramenta de produtividade** mas sim como **uma ferramenta exclusivamente para aprender e/ou ensinar a lógica de programação totalmente em português**, que permite escrever programas usando palavras chave em português, executar um programa passo a passo, visualizar o conteúdo das variáveis e *profiling* de um programa. Além disso a ferramenta permite converter o código fonte, escrito na nossa linguagem (em português) para a linguagem C e Java, possibilitando que o aprendiz do CompAlg possa desde cedo ter algum contato com estas duas linguagens mais usadas nas Universidades angolanas. A ferramenta disponibiliza ainda um manual completo de suporte a linguagem e de suporte a utilização da ferramenta, bem como uma ajuda online através do correio eletrónico.

Índice

Lista de Tabelas

Lista de Figuras

Lista de Exemplos

Introdução 1

Com o avanço tecnológico na era da informática o mercado de trabalho passou a exigir uma demanda maior de profissionais na área de computação, tanto engenheiros e cientistas da computação como também outros ramos de tecnologia da informação. Uma consequência dessa procura por pessoas especializadas na área é o crescente número de cursos ofertados nas instituições de ensino superior ligados ao setor computacional. Entretanto, apesar do progressivo incentivo no sentido de formar novos profissionais, é notório que existe alguma coisa dificultando atingir esse objetivo em grande escala. Tal facto deve-se ao baixo índice de conclusão dos alunos nos referidos cursos. Ou seja, observa-se um elevado número de reprovações e evasão de alunos matriculados nos cursos de computação em instituições de ensino superior. De acordo com Duarte et al. [Dua10], os cursos de computação são os de maior índice de evasão em instituições de ensino superior do Brasil, que chega a 28% do total de alunos matriculados.

Nesse sentido, muitas pesquisas têm sido realizadas na tentativa de entender e solucionar essa questão que aparentemente é um problema voltado para o processo de ensino-aprendizagem de lógica da programação. Em investigações voltadas aos cursos de computação em instituições de ensino superior, foi constatada uma alta taxa de reprovação nas disciplinas de algoritmo e programação. Para os autores Pereira Jr. et al. [Jún05] os grandes vilões dos cursos de computação são as disciplinas de algoritmos e programação, que são ministradas logo no início dos cursos, e acabam sendo mesmo como um gargalo onde a maioria dos alunos já reprovam ou, desestimulados, acabam por abandonar a área definitivamente. Também, a disciplina de lógica de programação é diagnosticada como a de maior índice de reprovação nos cursos de computação ([Cam09, Mar03a, dS06, Jún05]). A autora H. V. da Rocha [dR88], por sua vez, relata sobre o índice de desistência e reprovação nos cursos de lógica de programação, os quais iniciam com média de 60 alunos e em poucos meses atingem uma taxa de reprovação/evasão de aproximadamente 60%. Um índice ainda mais alarmante foi apontado por Duarte et al., segundo os autores no semestre de 2009, houve evasão de 70% dos alunos matriculados na disciplina de Introdução a Programação no curso de

Licenciatura em Ciência da Computação da Universidade Federal da Paraíba - Brasil, um contexto que se aplica de forma geral ao ensino superior de Angola, tanto no ensino privado como público.

Muito embora as disciplinas que envolvam algoritmo e lógica de programação possam ser vistas como "vilãs" por alguns estudantes, elas são de extrema importância para o desenvolvimento do aluno e não somente nos cursos de computação. A componente curricular de lógica de programação é considerada uma das mais importantes para os alunos dos cursos de Tecnologia da Informação e Comunicação (TIC), pois nessas disciplinas os estudantes desenvolvem habilidades necessárias para as disciplinas posteriores no decorrer do curso. Para Carvalho et al. [CRM15] a disciplina de lógica de programação é de extrema importância não somente para cursos de ensino superior de Computação, como também em cursos de Matemática e as Engenharias. E isso se deve ao fato de os conhecimentos adquiridos em lógica de programação serem pré-requisitos para disciplinas posteriores nos cursos citados.

Como já foi falado anteriormente, as disciplinas de lógica de programação e similares são ministradas nos estágios iniciais dos cursos, é nela que os estudantes têm seu primeiro contato com a linguagem dos computadores e a construção de códigos para resolução de problemas. De acordo com Jesus e Brito [DJ09], disciplinas como algoritmos e programação de computadores exigem dos alunos habilidades e competências no raciocínio lógico, resolução de problemas e capacidade de abstração da solução em representação de linguagem de programação. Outros autores relatam que o maior obstáculo encontrado pelos alunos ao estudar lógica de programação é a abstração dos problemas numa representação de linguagem computacional, ou seja, concepção do algoritmo [Gom00].

Campos [Cam10], realizou uma pesquisa com alunos de instituições de ensino superior, na qual ele contou com uma amostra de 60 alunos de 2 turmas diferentes. Na sua pesquisa o autor constatou que 65% dos estudantes têm dificuldades em construir a sequência lógica de algoritmos, mesmo que seja de baixa complexidade. De acordo com os seus resultados, provenientes de entrevista com os alunos, a maior dificuldade encontrada na disciplina de programação é a construção de soluções para os problemas propostos, ou seja, construção do algoritmo. Campos [Cam09] aponta as seguintes razões para essa dificuldade dos alunos em construir os algoritmos: 1. **o nível de detalhes que a solução deve ter** e 2. **a dificuldade dos discentes em associar os procedimentos estabelecidos na linguagem natural com os procedimentos da linguagem de programação**.

Ainda como os agentes motivacionais para o alto índice de reprovação e evasão nos cursos de programação os autores (Barcelos et al. [Bar09], Duarte et al. [Dua10] e [Cam10]) citam: 1. **dificuldade na compreensão dos conceitos abstratos** 2. **tempo precário para dedicação aos estudos** e 3. **dificuldade na compreensão da sintaxe e semântica da linguagem de programação**

Com base nos números estatísticos, anteriormente mencionados, podemos então afirmar que há um grave problema no processo de ensino-aprendizagem relacionado com disciplinas de programação. Tendo em vista essa questão, muitos trabalhos vêm sendo realizados, ao longo de anos, com o intuito de ajudar os alunos e professores no processo de ensino-aprendizagem das matérias relacionadas a programação. Enquanto uns pesquisadores dedicam-se a estudar formas de melhorar o processo por via de planos de ensino ou novas metodologias [DJ09, Cam10, Bar14, Gom00, Dua10] outros focam-se em criar ferramentas para atuar no processo como um agente condutor às novas habilidades requeridas para a construção de algoritmos [Sal10, Jún15, Mot08, And13, CRM15, Bar09].

Muitas ferramentas têm sido desenvolvidas com o objetivo de potencializar o rendimento dos estudantes nas disciplinas de lógica de programação. Entre as ferramentas desenvolvidas estão jogos, aplicativos de dispositivo móvel, websites e compiladores, que usam entre outras abordagens a animação de algoritmos e pseudo-linguagens, que é uma maneira genérica de escrever um código de programação e é mais acessível a iniciantes. Algumas dessas ferramentas são VisualG, LOGO, WebPortugol, Scratch, Alice, LogicBlocks, JavaTool, Verto e CP (Compilador Portugol) entre outras. No decorrer deste trabalho falaremos um pouco mais sobre essas ferramentas.

Nas disciplinas de lógica de programação os estudantes são estimulados a utilizar o raciocínio lógico. De acordo com Cormen et al. [Cor02] a maneira mais ágil de usar o raciocínio lógico é com a utilização da língua nativa que é a linguagem comum do indivíduo. Ainda, Barbosa [Bar14] relata sobre a influência da escolha da linguagem de programação na aprendizagem inicial de lógica de programação.

1.1 Motivação

A principal motivação para a criação do CompAlg foi a dificuldade enfrentada pelos estudantes, nativos de língua portuguesa, em aprender a implementar algoritmos com linguagens de programação de alto nível, que são escritas em inglês. Ou seja, as linguagens de programação de alto nível normalmente são escritas com símbolos ou palavras chave em inglês, com isso os estudantes, de língua materna portuguesa, ao

se depararem com as disciplinas de programação encontram um agente que dificulta ainda mais a sua aprendizagem, a complexidade de entender o inglês. O CompAlg tem como base a utilização do Portugol (ver Secção 2.3.1), que é uma pseudo-linguagem escrita em Português, e tem por objetivo principal ajudar estudantes a aprender a lógica de programação sem que eles tenham que se preocupar com a língua inglesa durante o processo. É bom salientar que esta ferramenta foi criada para ser utilizada em turmas iniciais de cursos de programação, antes que os estudantes tenham contato com qualquer linguagem de programação profissional.

As primeiras versões do CompAlg começaram a ser desenvolvidas em 2012, pelo autor deste trabalho. Trata-se de um ambiente de desenvolvimento de pequenos programas, que utiliza como base a pseudo-linguagem Portugol (ver Secção 2.3.1), criado para auxiliar no processo de ensino-aprendizagem de lógica de programação, atualmente em uso na Universidade Agostinho Neto (UAN) Angola, desde 2012 e no Instituto Superior de Tecnologias de Informação e Comunicação (ISUTIC), Angola, desde 2014, e uma versão personalizada em uso na Universidade das Ciências Informáticas (UCI), Cuba, desde 2015.

O CompAlg na época em que foi criado satisfez o objetivo proposto que era, em geral, permitir a escrita e execução algoritmos com palavras chave em Português. Contudo, na versão mais atualizada, até ao presente trabalho, ele resumia-se a uma ferramenta bastante simples e que não apresentava muito suporte didático aos estudantes. Uma das poucas ajudas que ele fornecia era a identificação da linha de possível erro no código e a saída dos resultados pela consola. Com o passar do tempo vieram a descobrir-se várias falhas ao nível da construção dos programas e pouca flexibilidade no uso de métodos (procedimentos/funções).

1.2 Objetivos e Contribuições

Tendo em vista a simplicidade e as deficiências em termos didáticos inerentes ao CompAlg, esse trabalho visa implementar melhorias nessa ferramenta de forma a deixa-lo mais robusto e atrativo para os estudantes. A ideia agora é transformar o CompAlg num ambiente de desenvolvimento integrado com características de um software educativo, dando aos estudantes a possibilidade visual e a interação com um plano de estudo que o estimule a seguir em frente no processo de aprendizagem.

Algumas das principais funcionalidades propostas são:

1. **Linguagem**

a) Assinaturas (procedimentos e funções).

b) *Break point* (instrução **parar**).

c) Função auto-completar para as palavras chave da linguagem.

d) Imprimir uma estrutura de dados (Vetor ou Registo).

2. **Ferramenta gráfica**

a) Visualizador do conteúdo de variáveis (Simples, Vetores e Registos.)

b) Visualizador do resultado booleano da avaliação condicional.

c) *Profiling* dos métodos e variáveis em uso do programa.

Para atingir os objetivos propostos neste trabalho, começamos por fazer um levantamento bibliográfico acerca de novas tecnologias para o processo de ensino-aprendizagem de lógica de programação. Em seguida foi realizada uma análise minuciosa sobre o atual *status* do CompAlg, no qual verificamos as falhas e possibilidades de implementar melhorias. Subsequentemente foram realizadas as novas implementações, as quais foram testadas e melhoradas tanto quanto necessário.

1.3 Estrutura do Documento

No decorrer deste documento falar-se-á sobre o desenvolvimento do trabalho. O texto encontra-se estruturado da seguinte forma:

Capítulo 1 - Introdução. Onde damos uma visão geral do problema abordado e dos objetivos inerente ao projeto.

Capítulo 2 - Sobre a Programação. Fazemos uma revisão sobre programação para auxiliar no entendimento de conceitos importantes utilizados para um melhor enquadramento do leitor ao tema abordado.

Capítulo 3 - Trabalhos Relacionados. Falamos sobre alguns trabalhos relacionados ao trabalho que pretendemos apresentar.

Capítulo 4 - A Ferramenta CompAlg. Apresentamos o nosso ambiente de desenvolvimento integrado, as partes que o envolvem e o processo de execução de cada uma dessas partes.

Capítulo 5 - O Processo de Compilação. Apresentamos todo detalhe do processo de compilação da nossa ferramenta e todas as partes que compõem esse processo.

Capítulo 6 - Funcionalidades da Ferramenta. Neste capítulo apresentamos todas as funcionalidades relevantes do nosso ambiente de desenvolvimento integrado.

Capítulo 7 - Conclusões e Perspetivas. Apresentamos as conclusões tiradas no nosso trabalho bem como o que se pode esperar num futuro próximo nas novas versões do CompAlg.

Sobre Programação 2

Entendemos que alguns conceitos são muito importantes quando abordamos sobre aprendizagem de programação, conceitos como lógica de programação, algoritmos, linguagem de programação, entre outros. Tendo em vista a uma melhor compreensão deste trabalho por parte do leitor, dedicamos este capítulo a uma breve revisão dos conceitos que julgamos relevantes. Assim, inicialmente apresentamos o conceito de lógica de programação, em seguida definimos o conceito de algoritmos, na Secção 2.3 abordamos um pouco sobre linguagem de programação, logo após discutimos um pouco sobre o cenário atual do ensino da lógica de programação.

2.1 Lógica de Programação

Antes de definirmos a **lógica de programação** é interessante entendermos o termo "lógica". No dicionário de língua portuguesa *priberam* a palavra "lógica" está definida como a ciência de raciocinar. E por outro lado a versão *dicionários modernos* imprensa pela Porto Editora (2016) define a "lógica" como estudo e determinação dos modos de pensamento discursivo que permitem evitar as contradições ou os erros. É através da lógica que conseguimos organizar os pensamentos de forma coerente, identificando o sentido das ideias e eliminando as possíveis falhas. De acordo com Forbellone et al. (2005) [For05] "a lógica é a arte de bem pensar", e é ela que coloca ordem no pensamento. Costuma-se frequentemente associar lógica apenas a matemática ou situações que envolvam cálculos exatos, contudo a lógica está ligada a todos os pensamentos e ações diárias, desde as mais simples até às mais complexas.

É usando a lógica que decidimos a ordem de execução de tarefas, como por exemplo, de uma forma bem simplificada, ao decidir escrever algo no seu computador sua lógica te indica um esquema a ser adotado que pode ser o seguinte: `sentar, ligar o computador, abrir o editor de texto e digitar`. É a lógica que orienta os passos a serem realizados para que o objetivo seja efetivamente alcançado.

É ainda através da lógica que verificamos a validade de argumentos. Forbellone et al.

(2005) [For05] utiliza o seguinte exemplo nesse sentido:

```
"Todo mamífero é um animal.  Todo cavalo é um mamífero.  Portanto,
                    todo cavalo é um animal.".
```

Exemplo 2.1: Exemplo que mostra uma frase lógica.

É através do uso da lógica que conseguimos verificar que essa sentença é verdadeira. Da mesma forma que o pensamento pode ser expresso em palavras, a lógica também o pode. Um exemplo disso é a "receita" já mencionada para se escrever algo no computador. Esta receita vem com uma sequência lógica de ações a serem executadas. É bom enfatizar que esta mesma sequência lógica pode ser escrita nos mais diversos idiomas existentes no mundo permanecendo intacta.

De forma a tornar nosso conceito mais completo, vamos definir a **programação**. Segundo o dicionário de língua portuguesa versão *dicionários modernos*, a "programação" é a elaboração de um programa de computador. Por outro lado na versão *priberam* a palavra "programação" aparece como a criação de um programa de computador. Deste modo, sempre que falarmos de "programação" estamos nos referindo a ação de escrever, criar ou elaborar um programa de computador com instruções bem definidas.

Finalmente chegamos a definição de **lógica de programação**, que é a sequência lógica utilizada com intuito de resolver problemas no âmbito de programação. É com a lógica de programação que definimos as etapas a serem seguidas rumo à melhor solução dos problemas a serem programados. Forbellone et al. (2005) [For05] define a lógica de programação como a ordem da razão e a comunicação formal para a programação de computadores. A lógica de programação pode também ser elaborada nas mais variadas linguagens de programação em estruturas conhecidas como algoritmos.

Até ao momento definimos a lógica de programação e inserimos dois conceitos novos: **algoritmos** e **linguagens de programação** que serão discutidas nas Secções que se seguem, respetivamente.

2.2 Algoritmos

A lógica de programação tem como principal objetivo a construção de algoritmos válidos. Mas o que vem a ser um algoritmo? Um **algoritmo** é a sequência de passos escritos, em uma determinada linguagem, para a solução de um dado problema. Forbellone et al. (2005) [For05] define **algoritmo** como "uma sequência de passos que visa atingir um objetivo bem definido". É comum referir-se a algoritmos também como

instruções ou comandos numa ordem lógica para resolver um problema previamente identificado.

Os autores Cormen et al. (2002) [Cor02] definem **algoritmo** como "uma sequência de passos computacionais que transformam a entrada na saída". Ou seja, o **algoritmo**, para a resolução de um dado problema, parte de dados ou informações de entrada e converte-os, após algum processo bem definido, em dados de saída que são as soluções para o problema proposto.

Um **algoritmo** pode ser `computacional` ou `não-computacional` dependendo da forma como este está representado/implementado. Quando nos referimos a algoritmos computacionais, estamos afirmando que o conjunto de instruções deste podem ser executadas pelo compilador através de um software específico, caso contrário dizemos que não é computacional. Para caso de estudo e perceção do nosso conceito vamos levar em consideração três formas de representação, isto é, **Fluxograma Convencional** em **Descrição Narrativa** e **Pseudo-linguagem**.

2.2.1 Descrição Narrativa (DN)

É uma das formas de representação textual de um algorítmo, segundo [For05], é a representação textual usando o português coloquial, mas sem "abusar" de toda riqueza gramatical da nossa língua pátria de formas a evitar a ambiguidade. Neste tipo de representação não estamos preocupados com palavras chave, nem de nenhuma estrutura em particular. O exemplo da Secção 2.1 para "escrever um texto em seu computador" está representado em descrição narrativa, e no entanto como não temos um conjunto de regras a seguir poderíamos resolver o mesmo problema usando outras sentenças mas que na mesma, resolve o problema proposto, por exemplo: • **Sentar, ligar o computador, abrir o editor de texto e digitar** ou • **Ligar o computador, abrir um editor de texto e escrever** ou • **Acomodar-se, se o computador estiver desligado ligar, abrir um editor de texto e escrever, senão, abrir um editor de texto e escrever**. Enfim, podemos utilizar as várias formas verbais que dão sentido para a solução que se pretende. Nesta representação estamos preocupados com a ordem em que as sentenças aparecem, por exemplo, não podemos escrever o texto antes de garantir que o computador está ligado, mas por outro lado podemos ligar o computador antes mesmo de nos acomodarmos. O Exemplo 2.2, permite-nos solucionar o problema da soma de dois números reais.

Como podemos notar no Exemplo 2.2, em cada linha temos uma ação que é concretizada pelo verbo no infinitivo que indica qual ação será processada sem quaisquer

1. Receber o primeiro número real
2. Receber o segundo número real
3. Efetuar a soma dos números recebidos
4. Apresentar o resultado.

Exemplo 2.2: Exemplo que mostra a soma de dois números em DN.

ambiguidade. São apenas algumas pequenas regras para que o algoritmo seja o mais claro possível.

2.2.2 Fluxograma Convencional

É a forma de representação gráfica de algoritmos. Forbellone et al.(2005) [For05] diz que, as formas gráficas são mais fieis ao raciocínio original, substituindo a grande quantidade de palavras por desenhos (figuras geométricas). Vimos que a representação em descrição narrativa não exigia regras ou conhecimentos de alguma estrutura em particular. A representação em fluxograma exige que tenhamos o conhecimento das convenções gráficas para representar as ações. Nesta forma de representação estamos preocupados apenas com a sequência lógica do problema em questão. Consideremos o exemplo apresentado na Figura 2.1:

No exemplo da Figura 2.1, cada figura geométrica dá-nos um indicador da ação que será executada. Assim, a figura para receber ou ler valores não é igual à figura para mostrar um resultado que por sua vez é diferente da figura para executar uma operação aritmética ou testar uma condição. Como já referimos anteriormente, neste tipo de representação estamos preocupados na sequência lógica do algoritmo.

2.2.3 Pseudo-linguagem ou Pseudocódigo

O Pseudocódigo tal como a representação em descrição narrativa é textual, porém não temos tanta liberdade de escrever as sentenças à nossa maneira. Segundo Ferrari e Cechinel (2008) [FER08] é uma maneira intermediária entre a linguagem natural (descrição narrativa) e uma linguagem de programação (C, Java, etc.) de representar um algoritmo. Nessa representação já temos um conjunto restrito de palavras chave normalmente na língua nativa do programador. A ideia desta forma de representar os algoritmos é a de familiarizar o aprendiz com algum formalismo embora sempre focado na lógica de programação de forma que seja muito mais fácil a integração em

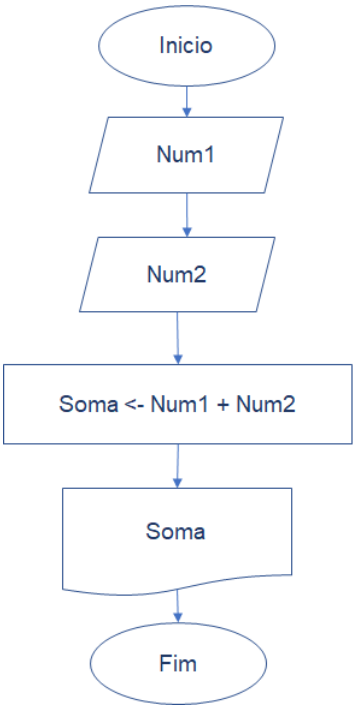


Figura 2.1: Fluxograma da soma de dois números reais.

linguagens de programação de alto nível. O Exemplo 2.2 que fizemos em Descrição Narrativa, pode ser feito em Pseudocódigo como mostramos abaixo.

1. a,b, soma: **Real**
2. **Ler** a,b
3. soma = a + b
4. **Imprima** soma

Exemplo 2.3: Exemplo que mostra a soma de dois números em Pseudocódigo.

Neste trabalho o nosso foco está assente nesta forma de representação como veremos na Secção seguinte sobre `linguagem de programação`.

2.3 Linguagem de Programação

O significado de `linguagem de programação` está amplamente associado ao conceito de linguagem, portanto, trataremos inicialmente da compreensão desse conceito. A `linguagem` pode ser definida como uma ferramenta utilizada para possibilitar a comunicação entre os indivíduos. O tipo de linguagem a ser utilizada é variável com a

situação ou meio onde se pretende comunicar determinada informação. Essas linguagens podem ser verbais ou não verbais, sendo que a verbal utiliza a fala e a escrita com esquemas organizados de palavras, e a não verbal pode utilizar desenhos, gestos, músicas e etc. Assim, quando estamos conversando pessoalmente com outras pessoas fazemos uso da linguagem verbal por meio da fala, e se a comunicação for através de correspondência fazemos uso da linguagem verbal escrita. O mesmo se aplica às pessoas que só podem se comunicar utilizando a linguagem gestual.

A `linguagem de programação`, por sua vez, é a ferramenta utilizada pelo programador para comunicar determinada informação ao computador por meio de um compilador ou interpretador. Ou seja, é um conjunto de informações textuais e simbólicas que podem ser utilizadas para instruir o computador a executar os procedimentos desejados. O autor Rafael Santos (2003) [San03] define linguagem de programação como "Uma ferramenta que determina as regras específicas e bem determinadas, com um conjunto de operadores e comandos, que podem ser usados para a construção de um algoritmo".

Ao construir-se um algoritmo, conhecer uma linguagem é tão importante quanto desenvolver o raciocínio lógico acerca do problema a ser solucionado. De acordo com Jesus (2009)[DJ09] para se construir um algoritmo é necessário se desenvolver um raciocínio lógico, é preciso compreender o funcionamento das linguagens de programação, quanto a sintaxe e semântica e só então será possível traduzir uma solução algorítmica para um programa de computador.

Da mesma forma que a linguagem humana possui diversos idiomas, como por exemplo o inglês, o francês, o japonês entre outras; a linguagem de programação também possui vários idiomas que se diferenciam pela sua sintaxe, semântica e paradigma (a forma de escrever o código ou resolver o problema proposto). Alguns exemplos de linguagens de programação são: Java, C, C#, C++, Python, Haskell, IDL, etc.

2.3.1 O Portugol

É uma pseudo-linguagem, criada pelos professores: António Carlos Nicolodi (Brasil) em 1983 e António Manso (Portugal) em 1986 foram os dois que praticamente iniciaram o seu uso, inicialmente foi muito influenciada pelo Pascal e pelo ALGOL. Hoje o Portugol tem sido muito influenciada por outras linguagens como C, C++ e Java.

O Portugol é bastante utilizado para ensinar lógica de programação e algoritmos. Está presente em muitos materiais didáticos de programação, e é todo escrito em português.

O Portugol está na base da linguagem suportada pelo CompAlg e outros compiladores e interpretadores como, o VisualG 3.0, Portugol Stúdio, G-Portugol, Portugol Viana. Reservamos o Capítulo 3 para falarmos sobre as ferramentas relacionamos e da influência que tiveram para o desenvolvimento do nosso trabalho.

2.4 Sobre o Ensino de Lógica de Programação

Como já foi mencionado em outro momento, que as disciplinas introdutórias de lógica de programação são, de certa forma, um pilar para o desenvolvimento dos estudantes que anseiam dominar a arte de programar computadores. A ferramenta utilizada no presente trabalho tem por objetivo auxiliar no processo de ensino e aprendizagem dessas disciplinas. A matéria de introdução a lógica de programação geralmente é oferecida no início de cursos de computação e também em cursos que exijam conhecimentos de programação de computadores, como por exemplo cursos da área de exatas e tecnologia da informação. Essas disciplinas podem vir com nomes diferentes como por exemplo: Técnicas de Linguagens de Programação, Lógica de Programação, Introdução à Programação, etc., porém o objetivo geral é sempre o mesmo: formar discentes capacitados a construir algoritmos eficientes para a resolução de problemas básicos, utilizando com destreza a lógica de programação ([For05, Cam09]).

Devido ao alto índice de reprovação e evasão nas disciplinas de introdução a programação, muitas pesquisas vêm sendo realizadas, ao longo dos anos, com o objetivo de potencializar o aprendizado dos estudantes nas referidas matérias. Nesse contexto, alguns autores dedicam-se a descobrir as principais dificuldades dos discentes e buscam formas de melhorar o processo de ensino-aprendizagem por meio de novas metodologias de ensino. Neste sentido, Delgado (2004) [Del04] sugere, na sua pesquisa acerca do processo de aprendizagem de lógica de programação, uma metodologia para o ensino em disciplinas introdutórias de programação e algoritmo. Na sua proposta ele descreve três fases as quais devem ser seguidas: 1. `Resolução de problemas`, 2. `Formalização, valorizando a concisão e a precisão da linguagem utilizada` e 3. `Construção de algoritmos`.

Esta metodologia foi testada em turmas de ensino superior e o autor relata que houve um rendimento mais satisfatório com o uso da metodologia, observando bons resultados e uma curva de aprendizagem mais suave.

Segundo Renumol at al. [Ren09], pressupõe que o maior e talvez o principal problema seja o `domínio cognitivo independente da linguagem de programação`, levando em consideração a *Taxonomia de Bloom* (conhecimento, compreensão, aplicação, análise

e avaliação), afirmando mesmo que se houver um trabalho árduo nessas habilidades os estudantes poderão melhorar significativamente a sua aprendizagem de programação. O mesmo afirma ainda que, o domínio cognitivo tem uma alta relevância no ensino e aprendizagem da programação em relação ao domínio psicomotor.

Em contrapartida Campos (2010) [Cam10] sugere e experimenta uma nova metodologia de ensino, conhecida como ERM2C (Entender, Revisar, Melhorar, Complementar e Construir). Esta inovadora metodologia baseia-se em cinco etapas que prometem guiar os estudantes num processo de ensino-aprendizagem mais consistente e satisfatório. Num primeiro momento o estudante é convidado a entender a formalização e o funcionamento de um algoritmo já pronto. Nesta etapa ele irá se familiarizar com a estrutura de um algoritmo não se preocupando em fazer mudanças ou corrigir nada. A etapa seguinte é a revisão. O estudante será estimulado a fazer uma revisão do código e procurar possíveis falhas ou formas de melhorá-lo. Logo após a revisão o estudante é convidado a fazer mudanças no programa de forma que ele se torne mais eficiente. Na quarta etapa será realizado um complemento no algoritmo que já existe, ou seja, o estudante terá de acrescentar alguma funcionalidade ao programa. E por fim, o discente deverá construir seu próprio algoritmo. Esta metodologia foi testada pelo autor do trabalho em turmas de ensino superior e os resultados foram satisfatórios, ao final dos semestres constatou-se que houve uma diminuição do número de reprovação dos alunos com o uso da metodologia.

Alguns outros trabalhos foram realizados nesse contexto, como por exemplo Fernandes (2002) [Fer02], que propõe um programa/plano para disciplinas introdutórias de programação para os cursos de computação, e Jesus e Brito (2009) [DJ09], que sugere um plano de ensino para a matéria. Também, Gomes (2000) [Gom00] sugere uma ferramenta para auxiliar no processo de ensino-aprendizagem, o Sistema Interactivo para Construção de Algoritmos e sua Simulação (SICAS). Consiste num ambiente criado para possibilitar aos alunos conceber, simular e testar algoritmos. Há também pesquisadores que defendem agentes comportamentais como possíveis auxiliares no processo de ensino-aprendizagem, o autor Duarte (2010) [Dua10] sugere a utilização da competitividade como um agente estimulante para o processo de ensino-aprendizagem.

Os trabalhos citados acima abordam a solução do problema do processo de ensino-aprendizagem de lógica de programação a partir da utilização de novos métodos ou planos de ensino. Em complemento a essa linha de pensamento, Barbosa et al. (2014) [Bar14] na sua pesquisa identificou uma dificuldade no processo de aprendizagem das disciplinas de introdução a programação. No seu trabalho foram realizadas análises com base no desempenho de turmas do ensino superior. Os resultados analisados

foram de turmas que aprenderam a programar utilizando a linguagem de programação Python ou C, ou seja algumas turmas num primeiro momento haviam sido estimuladas a programar em C, enquanto outras foram orientadas a programar em Python. A pesquisa analisou, se havia diferença no desempenho dos discentes para diferentes tipos de linguagem de programação. De acordo com os resultados de Barbosa et al. notou-se uma redução de 10% no total de reprovações nas turmas de alunos novatos que usaram Python como primeira linguagem. Sendo assim verifica-se que há sim uma relação entre o rendimento dos alunos e a linguagem de programação escolhida para as disciplinas introdutórias.

Na mesma linha de pensamento Mota e Pereira (2008) [Mot08] sugerem o uso de pseudocódigo para as disciplinas iniciais de programação. O pseudocódigo, se utilizado nessa etapa da aprendizagem, ajuda a minimizar as dificuldades dos estudantes devido a formalização de uma nova linguagem.

Ainda acerca da influência da linguagem de programação no processo de aprendizagem de programação, Martins e Correia (2003) [Mar03b] fizeram um estudo com base na utilização da linguagem *LOGO* por estudantes matriculados em cursos de ciências da computação. Neste estudo a linguagem *LOGO* foi empregada como metodologia de apoio ao desenvolvimento do raciocínio lógico. Os autores constataram que a linguagem *LOGO* pode ajudar os alunos na aprendizagem de lógica de programação.

Em contrapartida, muitos outros trabalhos vêm sendo realizados visando a construção de ferramentas como aplicativos, websites e jogos para serem utilizados no processo de aprendizagem de programação. Um bom exemplo desse tipo de ferramenta é o *JavaTool*, apresentado por Mota et al. (2008) [Mot08], que consiste num software cujo objetivo é auxiliar na aprendizagem de estudantes de cursos de programação em instituições de nível superior. O *JavaTool* traz opções de animação e execução de códigos, o que promete estimular os discentes em seu processo de aprendizagem. A linguagem de programação utilizada pelo *JavaTool* é uma versão simplificada da linguagem Java.

Por sua vez, Junior e Boniat (2015) [Jún15] desenvolveram um projeto em website, chamado de *LogicBlocks*, que, similarmente, objetiva otimizar a aprendizagem dos estudantes de lógica de programação. Esta ferramenta oferece uma forma de resolução de problemas de lógica por meio de utilização de blocos de encaixe, e desta forma o discente é levado a exercitar seu raciocínio lógico enquanto resolve determinados desafios propostos.

Um projecto interessante é o `CODEWITZ` [COD01c], uma rede internacional para melho-

rar a aprendizagem da programação, segundo o próprio site, o principal objetivo é planejar, produzir e avaliar auxílios únicos de ilustração, animação e visualização para estudantes e professores de programação de computadores. E segundo a equipa do projeto têm sido muito bem sucedido, porque, como já frisamos alguns dos problemas inerentes na aprendizagem da programação é exatamente a falta de ilustrações ou interações internas do problema que se está a resolver. E este projeto dedica-se inteiramente em criar ferramentas nesse sentido, todas na língua inglesa, desde 2001. Alguns sub-projetos foram criados com mesmo intuito tal como, `Codewitz-Minerva` [COD01a] e `Codewitz-Asialink` [COD01b] para citar alguns.

Há também alguns trabalhos que "apostaram" em jogos como ferramentas de apoio educacional, Sales e Dantas (2010) [Sal10] criaram um protótipo de jogo educativo para ajudar no processo de ensino-aprendizagem nas disciplinas de lógica de programação e correlatas. O jogo foi desenvolvido como um esquema de aventura onde o discente é estimulado a desenvolver habilidades de raciocínio lógico e resolução de problemas enquanto se diverte no desenrolar dos desafios. Um outro software com características similares de jogo é o *Alice*, desenvolvido em *Carnegie Mellon University* em 1997. O *Alice* é baseado em construção de algoritmos por meio de blocos e o discente pode criar por meio dele animações de jogos simples em 3D. A ideia principal do *Alice* é levar o estudante a desenvolver as habilidades inicias necessárias a um programador, com pensamento lógico e também levar os princípios fundamentais de programação orientada a objetos.

Além disso, também foram desenvolvidos aplicativos de dispositivos móveis para auxiliar na aprendizagem da programação, exemplos desses tipos de softwares são os *Blockly API, Blockly Games, Hour of Code e o APP Inventor*. Carvalho et al. (2015) [CRM15] fez uma análise do uso de aplicativos e hardwares como ferramenta de suporte ao ensino de programação. No seu trabalho Carvalho descreve a experiência realizada com estudantes, onde os discentes foram levados a utilizar um aplicativo ligado a um hardware (uma espécie de carrinho) que respondia aos comandos dos estudantes que criaram seus comandos em códigos de programação no aplicativo. Com este tipo de estrutura, o estudante é estimulado a construir sequências lógicas e pode visualiza-las por meio do hardware. De acordo com o Carvalho et al. (2015) os resultados da experiência foram satisfatórios e os discentes aprovaram a utilização da ferramenta.

Há ainda aqueles pesquisadores que empenharam-se no desenvolvimento de ferramentas que compilam ou interpretam os algoritmos tais como o `Portugol Studio`, `G-Portugol`, `VisualG`, `MACP` e `Portugol Viana` e `Verto`. Foram todas criadas com o mesmo objetivo comum, o de apoiar o processo de ensino-aprendizagem da progra-

mação. **O nosso trabalho encaixa-se nesse grupo, pois estamos incrementando uma ferramenta que auxilia no processo de aprendizagem da lógica de programação**. Devido à relevância dessas últimas ferramentas citadas, o capítulo 3 traz uma breve descrição e avaliação das mesmas.

2.5 Síntese

Neste capítulo falamos sobre os conceitos mais importantes e abrangentes do nosso trabalho. Começamos por definir a lógica de programação (ver Secção 2.1) e sua importância para o nosso trabalho, abriram-se dois temas que abordamos com algum detalhe, o conceito de algoritmos (ver Secção 2.2), como poderíamos representa-los e das linguagens de programação (ver Secção 2.3) onde introduzimos de forma breve o Portugol (ver Secção 2.3.1) e sua influência para o nosso trabalho. Posteriormente abrimos uma Secção onde nos dedicamos a falar sobre o ensino da programação (ver Secção 2.4), das várias pesquisas e metodologias que tem sido experimentadas para um maior aproveitamento no ensino da programação. Nesta "viagem" sobre o ensino da programação enquadramos o contexto do nosso trabalho que está voltado para ferramentas que auxiliam o ensino e aprendizagem da lógica de programação.

Trabalhos Relacionados 3

Neste Capítulo abordaremos as ferramentas Portugol Studio, VisualG, Portugol Viana e MACP. O software Verto, que foi citado no final da Secção 2.4 do Capítulo 2, é um interpretador de código que trabalha com a linguagem Verto. Todos os outros são voltados a pseudo-linguagem Portugol, como frisamos na Subsecção 2.3.1 quando introduzimos o Portugol. Estes são de maior interesse para nosso trabalho, pois a ferramenta que otimizamos foca no uso de pseudocódigo como auxílio para a aprendizagem da lógica de programação, tendo como público alvo estudantes nativos de língua portuguesa, principalmente dos Países Africanos de Língua Oficial Portuguesa (PALOP), que além da dificuldade com as formalizações da própria linguagem de programação, ainda se deparam com a dificuldade de entender a simbolização dessas linguagens em outro idioma, geralmente o inglês. Nossa ferramenta auxilia os estudantes a escreverem seus códigos em Portugol ou Português Estruturado.

3.1 VisualG

Segundo o site oficial [Ant08], o VisualG é um programa que permite criar, editar, interpretar e que também executa os algoritmos em português estruturado (portugol) como se fosse um "programa" normal de computador. Foi a primeira ferramenta que conhecida pelo autor deste documento. É sem dúvida uma ferramenta poderosa, baseada na linguagem Pascal, foi desenvolvida inicialmente pelo professor Cláudio Morgador de Souza.

Ainda segundo informações do site [Ant08], o VisualG possui algumas boas funcionalidades tais como, a simulação da "tela" do MS-DOS, visualização de variáveis, "breakpoints", recurso de ajuda ajuda online, exporta o algoritmo para um código similar em Pascal, impressão dos fontes e outras características que auxiliam no aprendizado das técnicas de programação. Foi propositadamente criado para ajudar aos alunos iniciantes em programação, mas pode ser utilizado por professores e outros autodidatas. É de notar que esta ferramenta tem uma preocupação em apresen-

tar o `conteúdo das variáveis` que simula a memória interna do programa e também `execução passo à passo do programa`. A Figura 3.1 mostra a tela principal desta ferramenta.

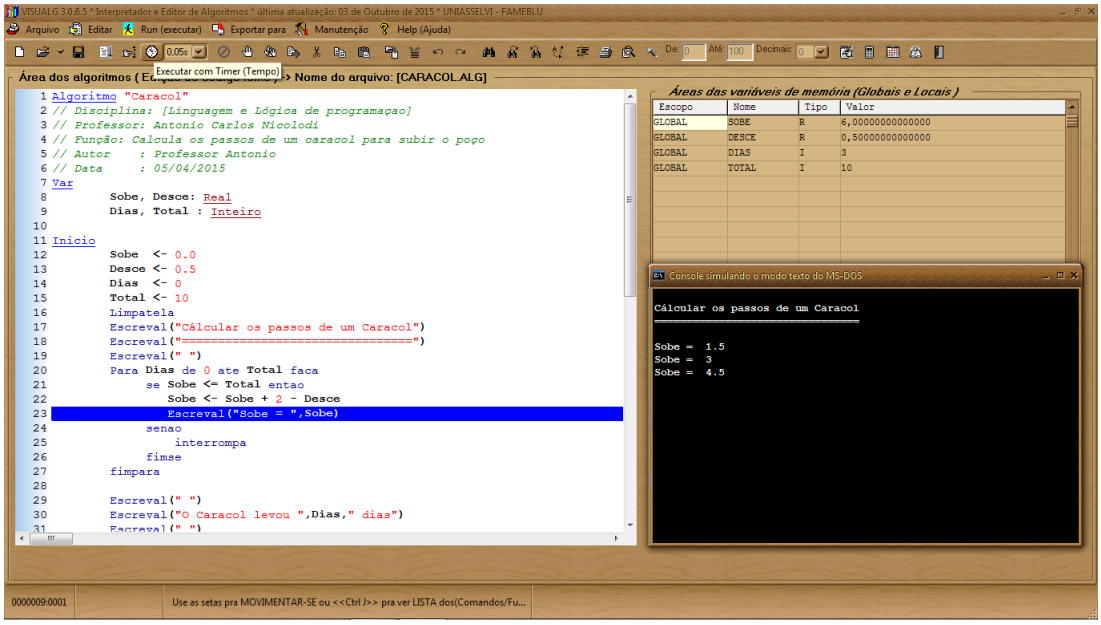


Figura 3.1: Tela principal do VisualG 3.0

A Figura 3.1 mostra a tela principal do VisualG com três janelas principais que são: esquerda (o editor de código); direita inferior (a consola de resultados); direita superior (o visualizador de conteúdos das variáveis).

Hoje, o projeto tem sido mantido pelo professor `Antônio Carlos Nicolodi`, como já foi dito anteriormente (ver Secção 2.3.1) é o criador do método de ensino de lógica de programação no aprendizado dos Algoritmos conhecido como Portugol (ver na mesma Secção 2.3.1).

3.2 Portugol Studio

Portugol Studio teve início em 2007 com o desenvolvimento do núcleo do Portugol Studio, feito pelo estudante Luiz Fernando Noschang na Universidade do Vale do Itajaí (UNIVALI), com a sua última versão lançada em 2013. Rapidamente tornou-se popular no Brasil, devido ao seu atrativo (*design*) e funcionalidades. Segundo Luis at. al. [Nos14], a interface do ambiente de programação e teste de programas está dividida em três regiões principais: (i) código Fonte, onde o estudante cria seu programa; (ii) árvore de símbolos, onde pode visualizar as variáveis do programa e (iii) console de Entrada e Saída/ Mensagens, onde visualiza mensagens de erro e onde

ocorre a interação via consola. Luis at. al. ainda salienta no seu artigo [Nos14] que, as **mensagens de erro** desta ferramenta foram cuidadosamente elaboradas no sentido de apresentar dicas úteis para um aluno iniciante.

Analisamos quatro pontos focos nesta ferramenta: (i) capacidade de suportar a construção de jogos; (ii) inspetor de variáveis; (iii) execução passo a passo; (iv) o seu design atrativo. A Figuras 3.2 e 3.3 apresentam a tela inicial e de edição de código do Portugol Studio, respetivamente.

Figura 3.2: Tela Inicial do Portugol Studio

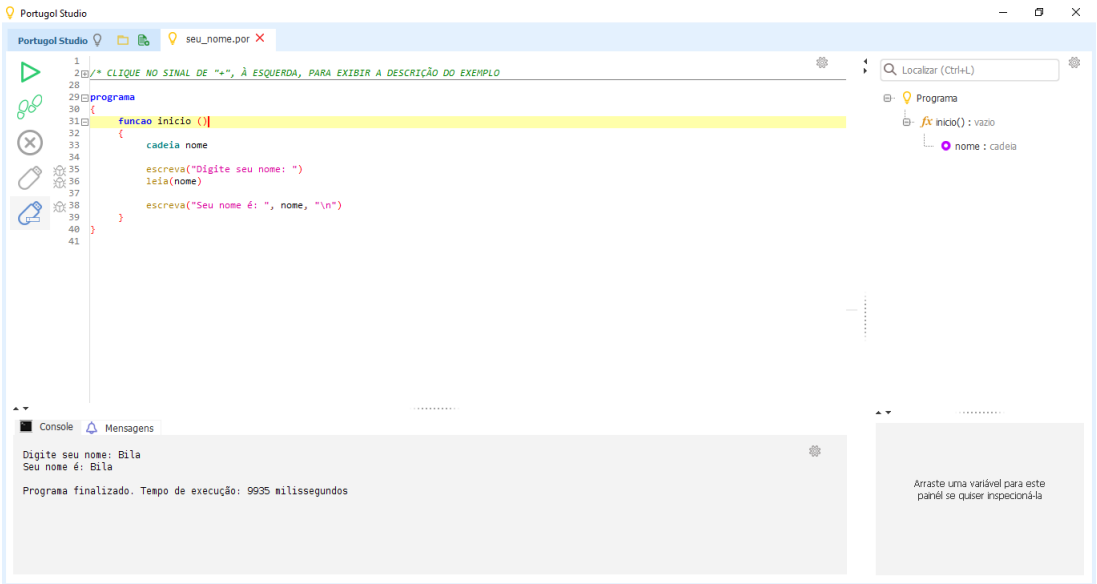


Figura 3.3: Tela de Edição do Portugol Studio

A Figura 3.2 mostra a tela inicial do Portugol Studio, com as várias opções e uma área de exemplos de algoritmos. A Figura 3.3 mostra a área reservada a codificação, onde apresentamos um exemplo de um pequeno programa que recebe o nome e apresenta-o na consola.

Não é muito difícil verificar que a versão do Portugol que esta ferramenta utiliza é ligeiramente diferente de tudo que falamos até aqui e essa diferença vai ser ainda mais evidente quando o leitor observar as outras ferramentas apresentadas nas Secções que se seguem. Isso porque o Portugol Studio utiliza a versão 2.0 do Portugol, que está muito baseada nas linguagens de programação C e PHP.

3.3 Portugol Viana

O Portugol Viana tal como o Portugol Studio e as outras ferramentas que abordamos nesta Secção, é *open-source* para edição/execução algorítmica. Portugol Viana permite a edição de algoritmos em português, tendo como base o Portugol (Secção 2.3.1). Nesta ferramenta os algoritmos são executados e monitorizados. A página oficial do Portugol Viana [VIA08] encontra-se desativada, mas alguma informação relativa à ferramenta bem como o ficheiro de instalação pode ser encontrado no *Sourceforge* [mig08]. Segundo esta mesma fonte a última atualização foi realizada em 2013.

Esta ferramenta foi desenvolvida em 2008, pela Escola Superior de Tecnologia e Gestão de Viana do Castelo. A ideia era criar um simulador de linguagem algorítmico para apoio às aulas de Introdução à Programação dos Cursos de Engenharia da Escola Superior de Tecnologia de Viana de Castelo mas tendo como ponto de partida uma versão já existente na Escola Superior de Tecnologia e Gestão de Tomar.

A versão 1.0 (atual versão) do Portugol Viana, para além da base, expande a linguagem Portugol "clássico" com funções, passagem de parâmetros por valor ou por referência, estruturas, listas ligadas, pilhas e filas. Foi também melhorada a interface com o utilizador de modo a possibilitar a animação de estruturas complexas (vetores, listas ligadas, pilhas e filas). Também apresenta os algoritmos em Fluxograma convencional (ver a Secção 2.2.2). Na figura 3.4 apresentamos a tela principal desta ferramenta com um programa em execução:

A Figura 3.4 apresenta um programa simples em Portugol, que recebe uma senha e verifica, informando ao utilizador que digitou a senha correta ou errada. A janela à direita mostra o mesmo algoritmo representado em Fluxograma convencional 2.2.2.

Até à última versão esta ferramenta era mantida por quatro programadores: Nelson

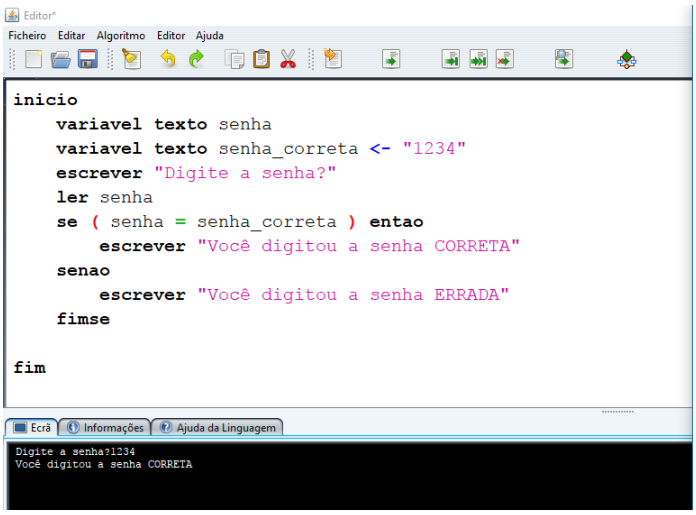


Figura 3.4: Tela Principal do Portugol Viana v.1.0, com um Programa em Execução.

Cerqueira - Desenvolvimento de novas funcionalidades e expansão da linguagem, Octávio Gradíssimo - Site do Projeto, Manuais de Utilizador (linguagem PortugolViana e ferramenta PortugolViana, Tiago Cunha - Correção de Erros e opção Multilingue e António Miguel Cruz - Coordenação do Projeto.

3.4 MACP - Compilador Portugol

Segundo o site oficial [MAC12], MACP é um projeto que foi idealizado e iniciado em 2006 na Universidade Federal do Maranhão. Foi desenvolvido durante a disciplina de Compiladores no curso de Graduação em Ciência da Computação pelos alunos na época: `Daniel Lima Gomes Júnior`, `Leandro de Sousa Marques` e `Ulysses Santos Sousa`.

Este projeto tem como objetivo principal `auxiliar estudantes brasileiros` aprendizes em atividades de programação, possibilitando que eles escrevam `algoritmos em Portugol` e possam gerar programas em `C, Pascal e Java` a partir do mesmo. Na figura 3.5 é apresentada a janela principal desta ferramenta.

A Figura 3.5 apresenta um programa semelhante ao da Secção 3.4, que recebe uma senha e verifica, informando ao utilizador que digitou a senha correta ou errada. E uma outra janela que apresenta a geração de um código equivalente ao algoritmo criado na linguagem C.

Desde o ano 2007 à 2012, este projeto foi mantido por dez pessoas das quais, dois professores coordenadores, cinco professores colaboradores e três alunos. Neste momento

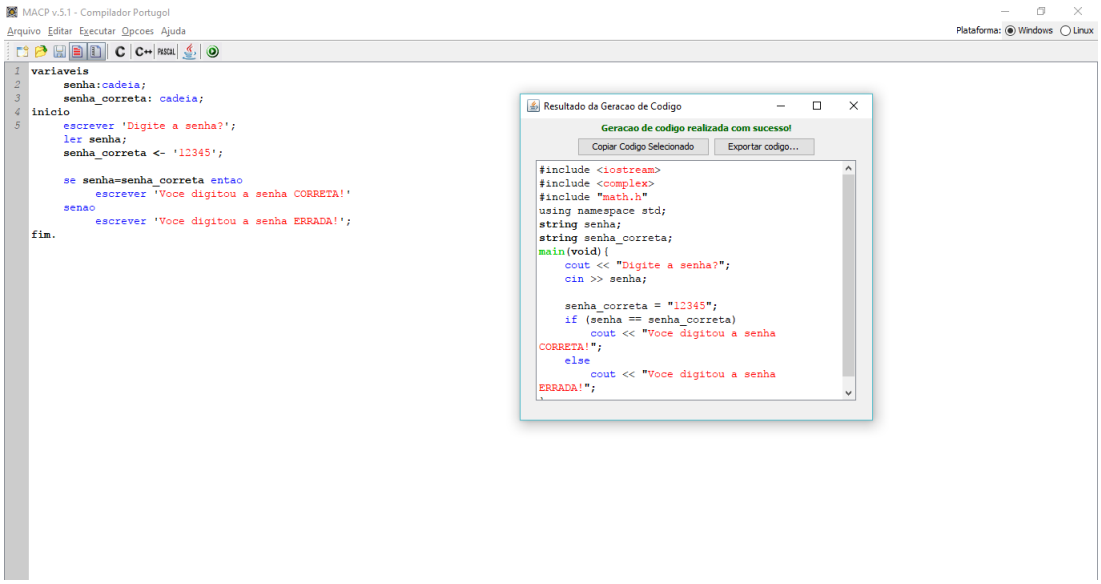


Figura 3.5: Tela Principal do MACP v5.1

o projeto está inativo desde 2012.

3.5 Avaliação

Nesta Secção nos dedicamos a fazer uma avaliação mais sintetizada e de uma forma comparativa das características das ferramentas estudadas e incluímos nesta análise a nossa ferramenta. A Tabela 3.1 mostra esta avaliação, que indica os pontos fortes e fracos de cada uma dessas:

Ferramenta	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)
VisualG			✓		✓		✓		✓	
Portugol Studio	✓		✓	✓					✓	✓
Portugol Viana	✓						✓		✓	
MACP	✓	✓					✓			
CompAlg	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Tabela 3.1: Comparação entre as Ferramentas de Ensino e Aprendizagem da Lógica de Programação

Legenda:

(i) Funciona no sistema *Windows, Linux e Mac OS.*

(ii) Possui conversor para a linguagem C e Java.

(iii) Possui a funcionalidade para visualizar conteúdos de variáveis.

(iv) Possui um inspetor de estruturas de dados.

(v) Possui *Breakpoint*.

(vi) Possui *Profiling* de métodos e variáveis.

(vii) Possibilita a ajuda *Online*.

(viii) Fornece um guia de estudo.

(ix) Permite a execução passo a passo.

(x) Possui um conjunto organizado de exemplos de programas.

A Tabela 3.1 mostra muito bem o estado atual das ferramentas destacadas nesse trabalho em relação à nossa ferramenta. Podemos observar que todas ferramentas estudadas com exceção o VisualG, foram projetadas para funcionar nos principais sistemas operativos (*Windows, Linux e Mac OS*), apesar disso o VisualG, tal como a nossa ferramenta são as únicas que implementam o *Breakpoint*. Destaca-se também a execução passo a passo que tem sido uma preocupação geral destas ferramentas com exceção da ferramenta `MACP`, que apesar disso é a única que faz conversão para a linguagem C e Java juntamente com a nossa ferramenta. As funcionalidades de *Profiling* e guia de estudo são novidades no desenvolvimento dessas ferramentas, sendo que apenas a nossa ferramenta possui tais possibilidades. Uma das grandes "falhas" é a não organização ou existência dos exemplos de programas por parte das ferramentas, e nesse ponto a nossa ferramenta e o Portugol Studio tiveram um cuidado especial de forma a proporcionar ao aprendiz uma sequência "perfeita" na sua aprendizagem.

3.6 Síntese

Neste capítulo focamos-nos principais trabalhos relacionados com o nosso, falamos com alguma brevidade sobre os grandes problemas inerentes ao ensino da programação em Angola e não só, e como essas ferramentas podem ser úteis para aprendizagem da lógica de programação. As ferramentas abordadas neste capítulo são: O VisualG, Portugol Studio, Portugol Viana e MACP-Compilador Portugol (Secções 3.1, 3.2, 3.3 e 3.4 respetivamente). Na parte final deste capítulo fizemos uma comparação entre as principais funcionalidades destas ferramentas e vimos como nossa ferramenta possui algumas vantagens em relação as outras. E com essa avaliação permitiu-nos observar também todas as funcionalidades que são comuns nesse tipo de projetos (ver a Tabela 3.1).

A Ferramenta CompAlg 4

Neste capítulo vamos apresentar a nossa ferramenta como ambiente de desenvolvimento integrado para aprendizagem da lógica de programação. Primeiramente apresentaremos uma breve história do nosso trabalho, para uma melhor compreensão da evolução desta ferramenta, em seguida uma visão geral das partes que constitui a ferramenta e posteriormente um detalhe pormenorizado da linguagem utilizada.

4.1 Um Breve Historial do CompAlg

O CompAlg, como já falamos anteriormente, é uma ferramenta desenvolvida para ser usada como apoio ao processo de ensino aprendizagem de disciplinas de lógica de programação e afins. Esta ferramenta foi desenvolvida em 2012 pelo autor desta dissertação, como um trabalho de conclusão do curso de Ciências da Computação na Faculdade de Ciências da Universidade Agostinho Neto (FCUAN) em Luanda, Angola. Embora o compilador em questão estivesse com suas funcionalidades operando satisfatoriamente, e tenha atingido os objetivos da época de sua criação, era bem percetível que ele poderia ser melhor aproveitado e mais eficiente no apoio aos estudantes, se fosse submetido a um novo projeto de implementação.

A versão *beta* foi lançada em 2012 e utilizada durante seis meses na FCUAN (período de testes, antes da apresentação ao público), por pelo menos 250 estudantes, na disciplina de **Programação I** do primeiro ano do Curso de Ciência da Computação desta instituição. Nesta altura vários erros foram encontrados pelos estudantes e docentes e corrigidos. Ainda no final do mesmo ano, foi lançada a versão 1.0 (versão estável, após a apresentação ao público no dia 5 Junho de 2012) com todas as correções e a possibilidade de utilização de métodos que retornam valores (inexistente na versão *beta*).

Na Figura 4.1 apresentamos a janela principal da versão 1.0 do CompAlg.

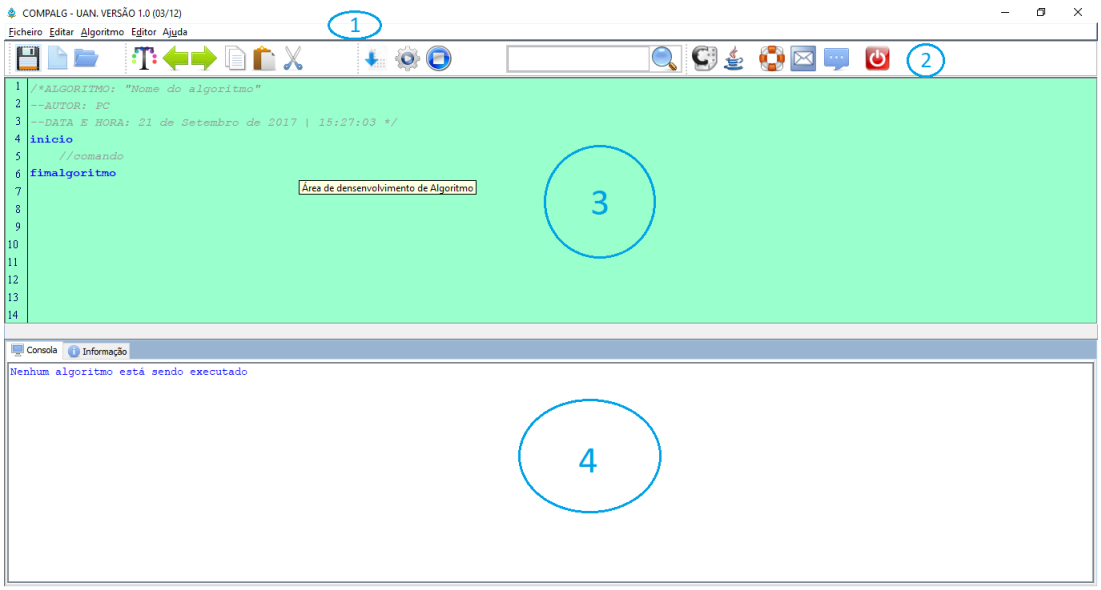


Figura 4.1: Janela Principal do CompAlg 1.0

Legenda:

1. Barra do menu principal.
2. Barra de tarefas (editor, programa, buscar, ajuda, conversor Java e C).
3. Editor de código.
4. Consola para apresentação de resultados e entrada de dados a partir do teclado e repórter de erros.

As Figuras 4.2 e 4.3 mostram o código em Portugol da Figura 4.1 convertido em C e Java respetivamente.

Legenda:

1. Opções para guardar, copiar, modificar ou cancelar.
2. Área do código convertido em C/Java.

A razão de incluir os conversores de código Portugol para as linguagens C e Java, reside no facto de que são as duas linguagens mais utilizadas nas universidades angolanas

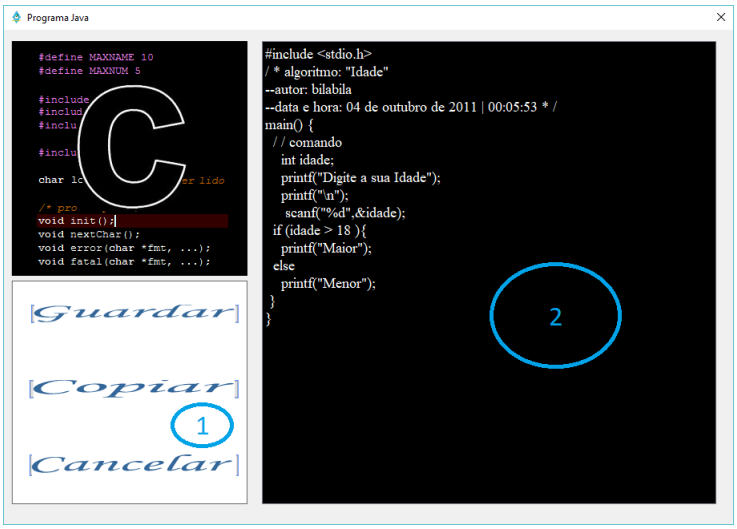


Figura 4.2: Conversor para C

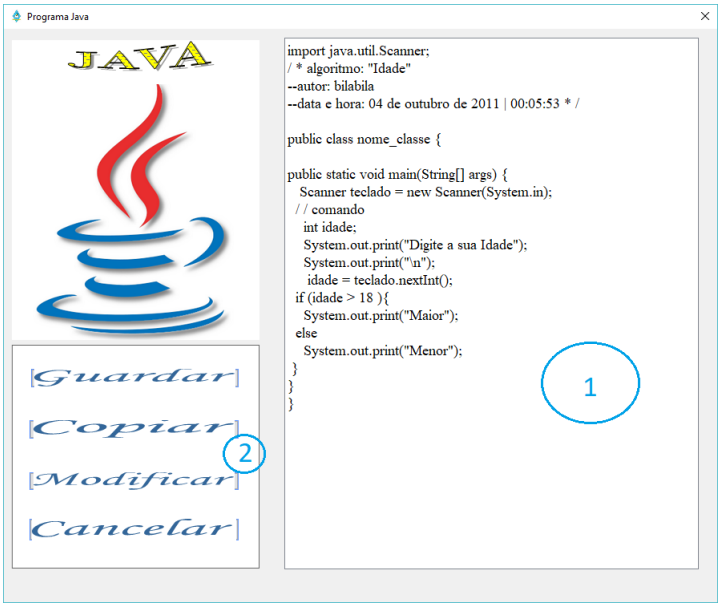


Figura 4.3: Conversor para Java

nas disciplinas que envolvem programação, principalmente nas instituições públicas do ensino superior.

Em 2014 foi lançada a versão 1.2 que entrou em uso no Instituto Superior de Tecnologias de Informação e Comunicação (ISUTIC), Luanda, Angola, com correções no funcionamento de registos e dos métodos. Nesta versão, já se teve em atenção ao *design* que na altura já se pretendia que fosse mais atrativo de modo a atrair o aprendiz da lógica de programação. No ISUTIC garantidamente, mais de 300 estudantes utilizaram o CompAlg como primeira ferramenta de aprendizagem da programação,

nos cursos de Engenharia Informática e Engenharia de Telecomunicações na disciplina de **Construção de Algoritmos e Programação (CAP)**.

Na Figura 4.4 apresentamos a tela principal desta versão.

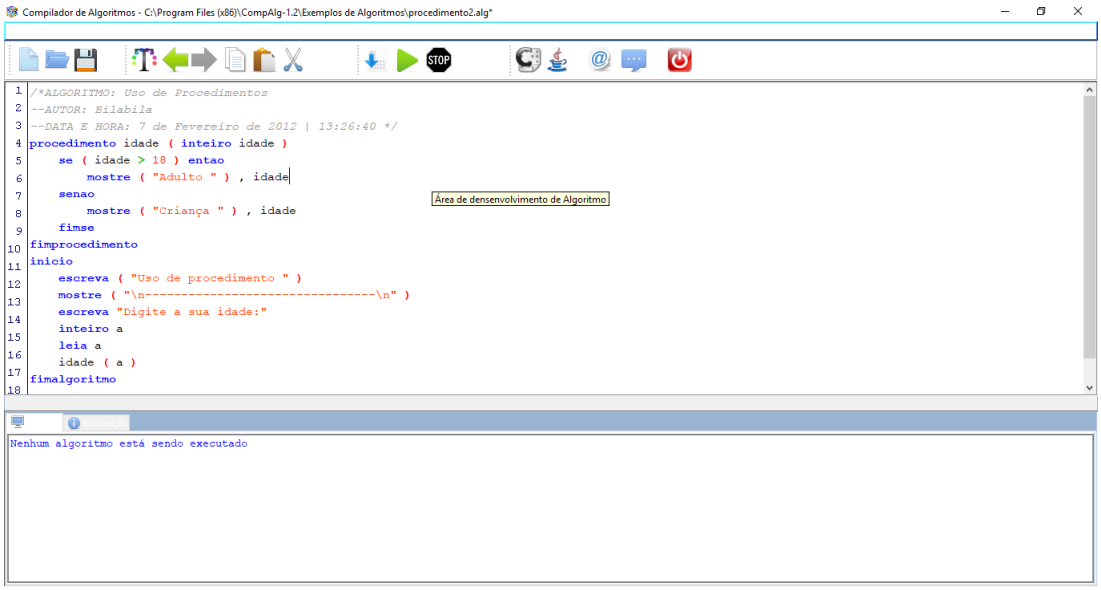


Figura 4.4: Janela Principal do CompAlg 1.2

Em 2015, uma última versão foi lançada antes da que pretendemos apresentar, a versão 2.0 em colaboração com David Barrera, Professor do ISUTIC, de nacionalidade cubana. Pretendia-se incluir um novo paradigma de programação (orientação a objetos). Esta versão embora muito instável, atingiu outros horizontes na sua utilização. A versão passou a ser adotada pela Universidade de Ciências Informáticas em Cuba por alguns estudantes, principalmente para os PALOPs que lá estudam nas disciplinas de lógica de programação. A Figura 4.5 mostra a tela principal desta versão.

Nesta versão do compAlg já há uma tentativa de execução passo a passo do programa, porém sem muito sucesso ao nível da implementação e visualização (*design*). Até ao momento vimos um pouco sobre a tragetória da nossa ferramenta desde 2012 até 2015. O sucesso que teve esta ferramenta (por ser a primeira em Angola) permitiu o desenvolvimento de alguns trabalhos relacionado a este em Angola tal como o `AngoCompiler` [Dev14] desenvolvido pelo estudante Adário Muatalembe, que afirma ter-se inspirado no CompAlg.

Reservamos a secção seguinte para vermos com algum pormenor o estado atual da nossa ferramenta.

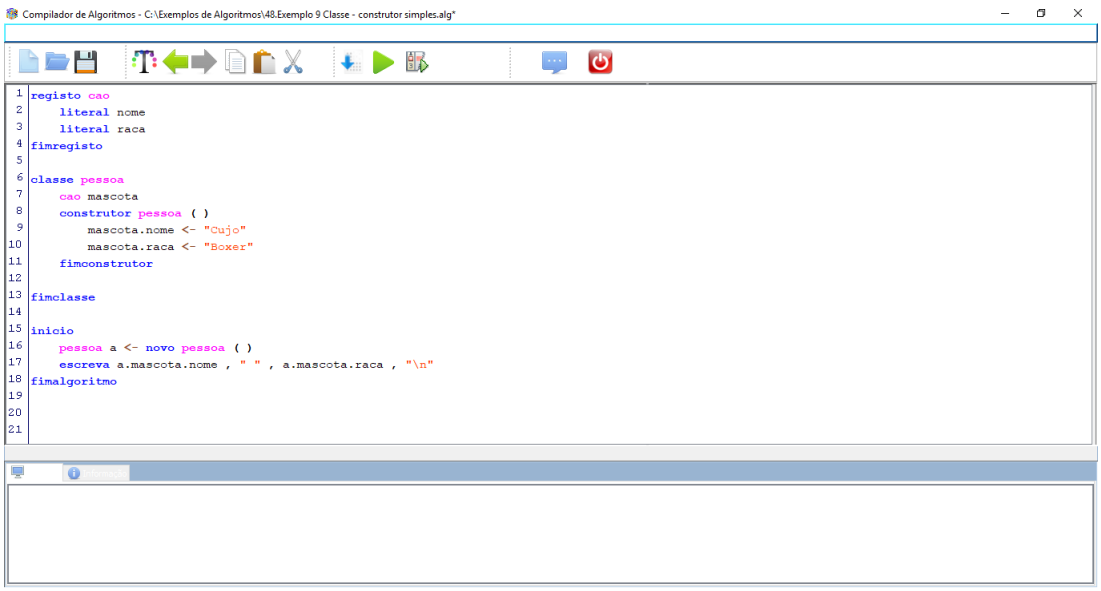


Figura 4.5: Janela Principal do CompAlg 2.0

4.2 Uma Visão Geral do Estado Atual

A nossa ferramenta é um ambiente de desenvolvimento integrado que permite: escrever código, executar, depurar, visualizar conteúdo de variáveis e obter o *profiling* de métodos e variáveis.

A Figura 4.6 mostra o nosso ambiente de desenvolvimento.

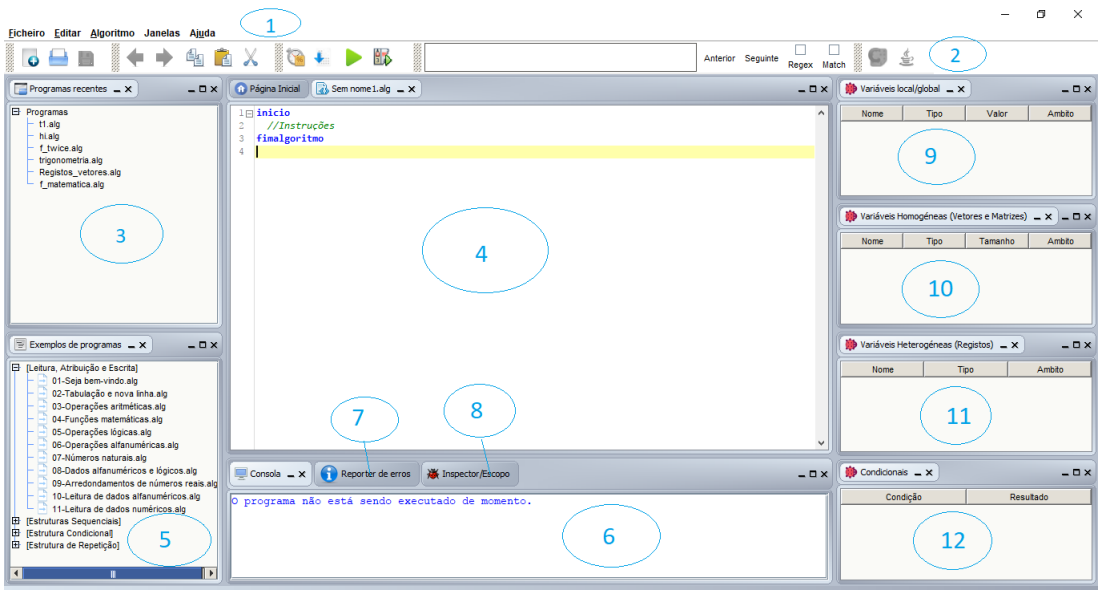


Figura 4.6: Nosso Ambiente de Desenvolvimento Integrado

Segue-se uma breve legenda da Figura 4.6 apresentada.

1. Barra do menu principal.
2. Barra de tarefas (editor, programa, buscar, conversor Java e C).
3. Programas recentemente abertos.
4. Editor de código.
5. Contém exemplos feitos de programas.
6. Consola para apresentação de resultados e entrada de dados a partir do teclado.
7. Repórter de erros durante a compilação ou a execução do programa.
8. Inspetor de estruturas de dados (Matrizes e Registos)
9. Visualizador de conteúdo das variáveis simples.
10. Visualizador de conteúdo de matrizes.
11. Visualizador de conteúdo dos registos.
12. Visualizador dos valores booleanos das condicionais do programa.

4.3 Metodologia

Analisaram-se cuidadosamente os detalhes da linguagem bem como os problemas de aprendizagem da lógica de programação por parte dos estudantes, que originou uma ferramenta totalmente dirigida a um grupo bem identificados de pessoas (ver Secção 1.1). Esta ferramenta foi inicialmente produzida para atender os problemas de aprendizagem da programação na Universidade Agostinho Neto, Angola. Foi implementada totalmente com a tecnologia *Swing* da linguagem Java, que possibilitou que a ferramenta funcionasse corretamente tanto no Sistema Operativo *Windows*, como também no *Linux* e *Mac OS*.

4.3.1 Ferramenta e Recursos Utilizados

A principal ferramenta utilizada para o desenvolvimento do CompAlg é a *IDE Netbeans*, tendo sido utilizada a versão 8 do Java. A escolha dessa ferramenta (*IDE Netbeans*) foi projetada desde o início do projeto (em 2012), por se apresentar bastante robusta e com muita flexibilidade no suporte de tecnologias Java, principalmente a *Framework Java Swing*, que foi utilizada para toda parte gráfica do nosso ambiente de desenvolvimento. Ainda sobre a parte gráfica, utilizamos um projeto de código

aberto ***RSyntaxTextArea*** [Dev12], que nos permitiu produzir um editor de código muito mais elegante e profissional. A nossa janela principal foi implementada com um projeto aberto da ***InfoNode*** [Inf98b], o ***Docking Windows***, cuja a documentação pode ser encontrada no próprio site oficial [Inf98a].

4.3.2 Pacotes e Classes

Nesta secção vamos-nos dedicar a apresentar apenas os pacotes, classes e métodos mais importantes da nossa ferramenta. Destacamos três pacotes essenciais da nossa ferramenta: **Editor**, **Portugol** e **Conversor**. A Figura 4.7 mostra como esses pacotes interagem entre si.

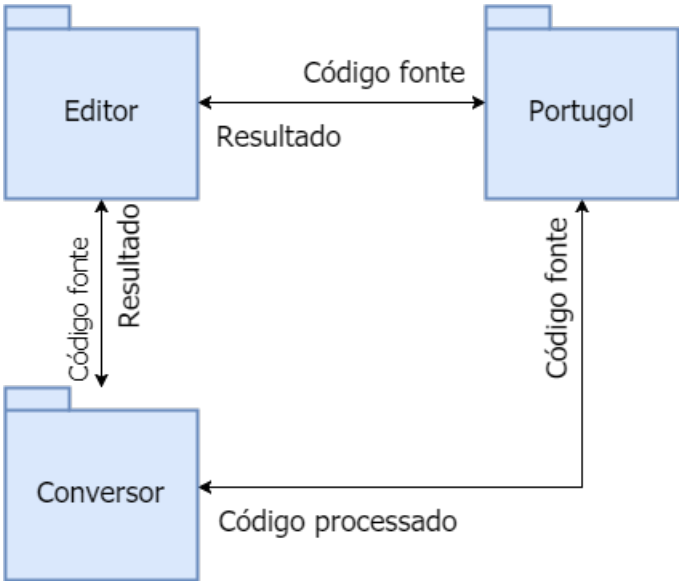


Figura 4.7: Interação entre os Principais Pacotes do Projeto

Podemos observar na Figura 4.7 que o pacote Editor (responsável pela nossa parte gráfica) disponibiliza o código fonte para o pacote Portugol e Conversor que depois de processar envia um resultado para o Editor. O Conversor é um módulo particular que recebe o código fonte do Editor por sua vez, e envia ao pacote Portugol, que devolve o código processado (passa pelo analisador léxico, sintático e semântico), que posteriormente é convertido em C ou Java na classe Conversor.

A Tabela 4.1 apresenta uma descrição resumida das funções de cada um desses pacotes. Cada pacote possui vários sub-pacotes e várias classes com funcionalidades distintas. A tabela 4.2, Apresenta uma breve descrição dessas classes.

Nome do pacote	Descrição
Editor	Este pacote contém 4 sub-pacotes e várias classes responsáveis por: 1. Criar o desenho gráfico da ferramenta; 2. Disponibilizar a ajuda da linguagem e da ferramenta graficamente; 3. Disponibilizar funcionalidades gráficas úteis para a janela principal, tais como: visualizador de conteúdo de variáveis, calendário, página inicial da ferramenta.
Portugol	É o pacote responsável pela linguagem, onde ocorre toda transformação do código fonte escrito através do editor. Esse pacote contém: 1. O nosso analisador léxico; 2. O nosso construtor sintático; 3. Analisador semântico; 4. A nossa consola que disponibiliza o resultado final do processamento.
Conversor	Um pacote auxiliar que trata da conversão na linguagem C e Java do código fonte escrito através do editor e compilado pelo nosso pacote da linguagem (Portugol).

Tabela 4.1: Principais Pacotes da Ferramenta.

Classe (.java)	Descrição
Intermediário	Classe do pacote Portugol, responsável por obter o código fonte e separar em "tokens", e que com ajuda de outras classes valida as instruções e constrói uma árvore sintática, que posteriormente é passada para análise semântica. Esta classe trata de passar o resultado na consola, caso não haja nenhum erro nas etapas salientadas anteriormente.
SubrotinaPlayer	Classe do pacote Portugol, muito semelhante a classe anterior mas, esta classe tem como objetivo fundamental gerir a execução passo a passo do programa. Esta classe permite "congelar" a execução a meio. Tem uma interação direta com a classe Intermediario.java.
LanguageException	Classe do pacote Portugol, responsável pela geração de erros e sugestão de possível solução. É invocada em qualquer uma das etapas (análise léxica, sintática ou semântica).
Editor_CompalgUP	Classe principal (JFrame) do pacote Editor, responsável pelo ambiente gráfico da ferramenta. Usa a biblioteca javax.swing para implementação de componentes gráficos, e implementa as classes: Runnable, RunListener, ActionListener.
construirProgramaC	Classe do pacote Conversor, responsável por converter um código fonte escrito na linguagem suportada pela ferramenta para a linguagem C.
construirProgramaJava	Classe do pacote Conversor, responsável por converter um código fonte escrito na linguagem suportada pela ferramenta para a linguagem Java.

Tabela 4.2: Descrição das Principais Classes.

4.3.3 Fluxo de Informação

A Figura 4.8 mostra o fluxo de informação a partir da classe principal `Editor_CompalgUP`. De salientar que apresentamos apenas as classes mais relevantes da nossa ferramenta. Como podemos observar na Figura 4.8, as classes `Intermediario` e `SubrotinaPlayer` são as classes mais importantes na interação entre o compilador da linguagem e a parte gráfica da nossa ferramenta.

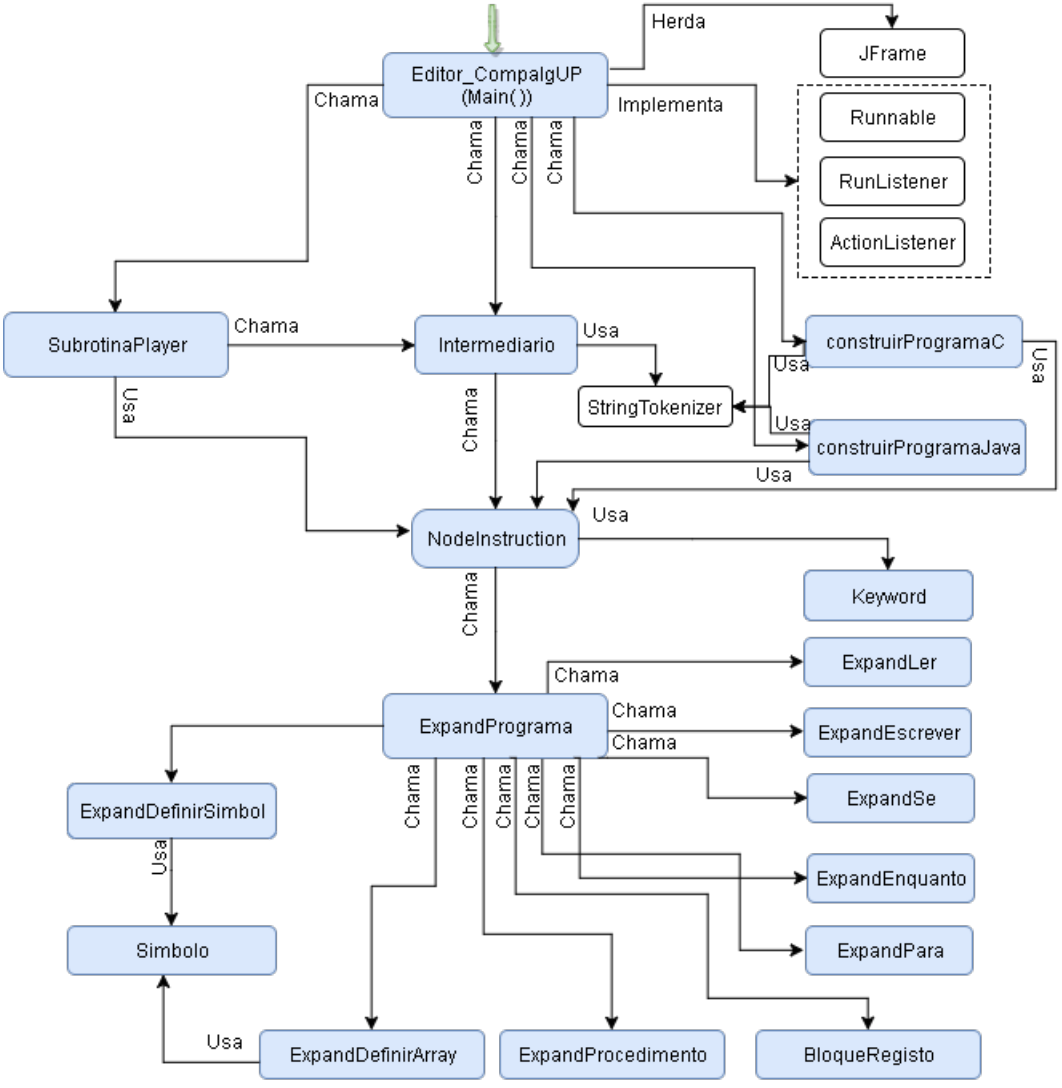


Figura 4.8: Fluxo de Informação do CompAlg

Como podemos observar na Figura 4.8, a classe **Editor_CompalgUP** desempenha um papel importante na interação com o utilizador, sendo responsável por obter o código fonte e posterior envio à classe **Intermediario** ou **SubrotinaPlayer**, dependendo da ação que o utilizador invocar. A classe **Intermediario** é útil para fazer a análise léxica do código fonte que em conjunto com as classes **NodeInstruction** e

ExpandPrograma permite construir a árvore sintática e fazer a análise semântica. A classe **SubrotinaPlayer** é útil principalmente quando a ação requisitada pelo utilizador é a execução passo a passo ou *profiling* de métodos e variáveis, sendo que aproveita as implementações produzidas nas classes **Intermediario** e **NodeInstruction** para a verificação da correção do código fonte e a consequente geração do resultado. O processo de compilação está detalhado no Capítulo 5.

4.4 A Linguagem de Programação

Como já foi introduzido na Subsecção 2.3.1 da Secção 2.3, quando falamos sobre linguagem de programação, afirmamos que o CompAlg tem como base a pseudo-linguagem Portugol (ver Secção 2.3.1), porém não tem evoluído como uma linguagem de programação normal, e ao longo dos anos alguns autores têm criado algumas poucas palavras chave. Sendo uma pseudo-linguagem abre-se uma liberdade aos criadores deste tipo de ferramentas para modificar ou estender a linguagem dependendo do ambiente ou da realidade do público alvo que se quer atingir. Por exemplo, na definição original do Portugol a instrução de saída de dados é o **escrever** e para entrada de dados utiliza-se o comando **ler**, em contrapartida no CompAlg usamos os comandos **escreva** e **leia**, respetivamente como veremos na subsecção dedicada á definição destas instruções (ver a Subsecção 4.4.1.9). Alguns novos comandos foram introduzidas na linguagem tais como: `assinatura` e `parar` que falaremos com detalhes ao longo da Secção 4.4.1.

4.4.1 Definição da Linguagem

Reservamos esta secção para apresentar com detalhes a definição da nossa linguagem (palavras chave e sintaxe). É de salientar que a nossa linguagem **não é sensível** a determinados casos ou seja para nossa linguagem é igual escrever "SE", "Se" ou "se".

4.4.1.1 Palavras Reservadas

A nossa linguagem possui atualmente 82 palavras chave, divididas em grupos de: tipos de dados, construção de métodos, comandos de entrada e saída de dados, construção de estruturas de controlo, alguns operadores aritméticos especiais e funções predefinidas da linguagem. As palavras reservadas da linguagem são: • assinatura • ate • caso • classe • constante • construtor • de • defeito • enquanto • entao • escolha • escreva • faca • ficheiro • fimalgoritmo • fimclasse • fimconstrutor • fimenquanto • fimescolha • fimfuncao • fimpara • fimprocedimento • fimregisto • fimse • funcao • inicio • leia

• mostre • novo • para • passo • procedimento • receba • registo • repita • retorne • se • senao • variavel • vazio • classe • caracter • inteiro • literal • logico • real • string • texto • vazio • abs • acoseno • acotangente • aleatorio • arredondar • aseno • atagente • comprimento • coseno • cosenoh • cotangente • cotangenteh • exp • limpatela • parar • ln • log • maior • menor • partefrac • parteinteira • potencia • raiz • seno • senoh • tangente • tangenteh • xou • div • e • nao • mod • ou.

4.4.1.2 Tipos de Dados

Como o objetivo da nossa pseudo-linguagem aqui definida é o ensino da lógica de programação, os tipos básicos serão os mais restritos possíveis, a fim de promover a independência da linguagem e da máquina. Os tipos de dados aceites são: inteiro, real, caracter, literal e lógico.

- **Inteiro**: Toda e qualquer informação numérica que pertence ao conjunto dos números inteiros relativos (negativo, nulo ou positivo), definidos com 32 bits, com valores entre -2147483648 á 2147483647. Por exemplo:
- **Real**: Toda e qualquer informação numérica que pertença ao conjunto dos números reais (negativo, nulo ou positivo), definidos com 64 bits, com valores entre -1.7E308 á 1.7E308. Por exemplo:

18.3; 0.410; 40.0; 1.1; 3.9; 0.74; 25; 354.0; 33.456334

Exemplo 4.1: Exemplo de Valores do Tipo Real.

- **Caracter**: Toda e qualquer informação composta por um caracter alfanumérico ou especial (Caracteres da Tabela ASCII) com valores de ASCII (0) á ASCII (255). Por exemplo:

"M"; "F"; "("; "B"; "-"

Exemplo 4.2: Exemplo de Valores do Tipo Caracter.

- **Literal**: Toda e qualquer informação composta por um conjunto de caracter alfanumérico e/ou especiais. Por exemplo:

"Marta"; "Festa"; "E eu (poder ser)"; "Bila"; "Rízia"

Exemplo 4.3: Exemplo de Valores do Tipo Literal.

- **Lógico:**: Toda e qualquer informação que pode assumir apenas um de dois valores possíveis (Verdadeiro ou Falso), definidos com 1 bit. Por exemplo:

"verdadeiro"; "falso"

Exemplo 4.4: Exemplo de Valores do Tipo Lógico.

4.4.1.3 Variável

Uma variável é um espaço reservado na memória do computador para armazenar um valor de um determinado tipo de dados (ver Subsecção 4.4.1.2). Variáveis devem receber nomes para poderem ser referenciadas e modificadas quando necessário. Algumas regras devem ser cumpridas para definir o nome de uma variável aceite pela nossa linguagem, são elas:

- O nome de uma variável deve começar sempre com uma letra ou *underline* (_).
- O nome da variável não pode conter espaços.
- Não deve conter caracteres especiais.
- Não pode ser uma palavra chave da nossa linguagem (ver sub-seção 4.4.1.1).

Nomes válidos: Bila, x, B5, nota1, media, xpto, _cool.

Nomes inválidos: 5x, a (19), a-b, nota/2

Sintaxe: **tipo de dado** <nome da variável>

Por Exemplo:

inteiro v1, v2, v3, s real num literal nome, morada

Exemplo 4.5: Exemplo que Mostra como Declarar uma Variável.

4.4.1.4 Estruturas de Dados

Atualmente a nossa linguagem suporta dois tipos de estruturas de dados que são: Vetores/Matrizes e Registo.

- **Vetores/Matrizes**: é uma coleção de variáveis do mesmo tipo, acessíveis com um único nome e armazenadas contiguamente na memória. O vetor é uma matriz de uma só dimensão. A primeira posição de um vetor é sempre o índice **0** e o último elemento encontra-se na posição do **tamanho do vetor - 1**. Por exemplo:

Inteiro a [10] - Cria um vetor 10x1, onde o primeiro elemento estaria em a [0] e o último em a [9].
Logico b [2][2] - Cria uma matriz 3x3, onde o primeiro elemento estaria em b [0][0] e o último em b [1][1].

Exemplo 4.6: Exemplo que Mostra como Declarar uma Matriz Uni-dimensional e outra Bi-dimensional.

- **Registo**: São definidas pela palavra chave **registo** e são compostas por campos (normalmente variáveis simples, vetores ou mesmo registos).

  Sintaxe:

  Registo <nome>

  <campos>

  fimregisto

  Por Exemplo:

```
Registo nome_registo
    inteiro Dia[2]
    literal Nome
    logico Matriculado
Fimregisto
```

Exemplo 4.7: Exemplo de um Registo com Três Campos.

No Exemplo, 4.7 definimos um registo com três campos. Para aceder aos campos desta estrutura basta primeiro cria um **identificador** do tipo do registo e depois usar o ponto "." para aceder qualquer campo. Por Exemplo:

```
nome_registo R1
R1.Nome ← "Bila"
inteiro dia1 ← R1.Dia[0]
```

Exemplo 4.8: Acesso aos Campos do Registo.

4.4.1.5 Atribuição

A atribuição de valores é representada pelo símbolo ←. Só é permitida a atribuição de um valor de cada vez. Uma atribuição válida possui um identificador válido à esquerda e uma expressão do mesmo tipo do identificador a direita.

Por Exemplo:

```
real nota1, nota2
nota1 ← 15.5
nota2 ← 18.5
```

Exemplo 4.9: Operador de Atribuição da Linguagem.

No Exemplo 4.9 atribuímos valores reais às variáveis definidas com o tipo de dado **real** **nota1** e **nota2**.

4.4.1.6 Operadores

Nesta Secção é apresentamos todos os operadores utilizados na nossa linguagem: Operadores Aritméticos, Lógicos e Relacionais.

- Operadores Aritméticos

Operação	Símbolo
Adição	+
Subtração	-
Divisão	/
Multiplicação	*
Resto da divisão	mod
Divisão inteira	div
Potênciação	^

Tabela 4.3: Operadores Aritméticos

- Operadores Lógicos

Operação	Símbolo
Inversão - não	não
Conjunção	e
Disjunção	ou

Tabela 4.4: Operadores Lógicos

- Operadores Relacionais

Operação	Símbolo
Maior	>
Menor	<
Maior ou Igual	>=
Diferença	<>

Tabela 4.5: Operadores Relacionais

4.4.1.7 Comentários

O comentário no código é uma forma de clarificar as instruções tanto para quem programa como para a pessoa que vai analisar o código. Portanto para tal é necessário fazer o uso do "//" para comentar em uma única linha e /* */ para comentar várias linhas.

Por Exemplo:

```
/*
* 03
* Programa: Funções matemáticas.alg
* Autor: Augusto Bilabila
* Ano: 2017
* Descrição:
* - O programa tem como objetivo principal calcular:
* 3 levantado a 5;
* Raiz quadrada de 125;
* Obter a parte inteira de um número real
* Seno de 30 graus.
*/
//Compilador de Algoritmos
```

Exemplo 4.10: Exemplo de Comentário na nossa Linguagem.

4.4.1.8 Corpo de um Programa

A estrutura de um programa aceite pelo nosso ambiente é de simples compreensão, os comandos que serão executados devem estar entre as instruções `inicio` e `fimalgoritmo`, com exceção dos registos e os métodos que devem ser criados antes da instrução `inicio`.

Sintaxe:

```
inicio

    //Instruções

fimalgoritmo
```

4.4.1.9 Comandos de Entrada e Saída de Dados

Os comandos que permitem a entrada manual via teclado e exibição de informações na tela são as instruções **leia** e **escreva** respetivamente.

- **Leia** : Comando de entrada de dados através do teclado. Exemplo: Por Exemplo:

leia (a) ou **leia** a

Exemplo 4.11: Exemplo de Leitura de um Dado através do Teclado.

- **Escreva** : Comando de saída que exibe os resultados do processamento do programa na consola. Por Exemplo:

escreva (a) **escreva** "Olá DCC"

Exemplo 4.12: Exemplo de Saída de Dados na Consola.

4.4.1.10 Estruturas de Controlo

São blocos que permitem controlar o programa, em tomadas de decisões ou repetições de instruções. Portanto as estruturas de controlo podem ser condicionais e de repetição.

1. **Estruturas Condicionais** - São comandos que avaliam o estado de uma condição para execução de uma determinada instrução. Na linguagem da nossa ferramenta consideramos dois tipos desta estrutura: o comando "se" e o "escolha".

- O comando **se**: avalia uma condição ou expressão condicional, que obrigatoriamente retorna um valor lógico (Verdadeiro ou Falso). Dependendo deste valor a execução passa para o bloco **então** ou **senão**. A estrutura de decisão pode ser Simples ou Composta, baseada num resultado lógico.

  Sintaxe:

  Estrutura Simples:

  se (condição) entao
  comando1
  fimse

  Estrutura Composta:

  se (condição) entao
  comando 1
  senao
  comando 2
  fimse

  Estrutura Composta com bloco:

  se (condição 1) entao
  comando1
  comandoN
  senao
  se (condição 2) entao
  comando 1
  comando N
  fimse fimse

  Por Exemplo:

```
se (idade >18) entao
   escreva "Maior"
senao
   escreva "Menor"
fimse
```

Exemplo 4.13: Exemplo de uma Estrutura Condicional "SE".

- A estrutura do tipo **escolha** seleciona uma determinada alternativa de uma única variável ou seja verifica se uma determinada variável possui um determinado valor num determinado instante.

Sintaxe:

Escolha (variavel)

Caso expressão1:

Comando 1

Caso expressão2:

Comando 2

Caso expressão3:

Comando 3

Defeito:

Comando N

fimescolha

Observação: A cláusula **Defeito** é executada quando nenhum dos casos se verifica. Por Exemplo:

```
Escolha (sexo)
    Caso "F":
        Escreva "Grande Mulher"
    Caso "M":
        Escreva "Grande Homem"
    Defeito:
        Escreva "Opção inválida!"
fimescolha
```

Exemplo 4.14: Exemplo de uma Estrutura Condicional "ESCOLHA".

No Exemplo 4.17, a variável de escolha é o "sexo". Se o conteúdo desta variável for "F" então deverá imprimir na tela "Grande mulher", se o conteúdo for "M" então imprimirá "Grande Homem" mas caso ele assumir um valor diferente de "F" ou "M" então imprimirá na tela a mensagem "Opção inválida!" .

2. Estruturas de **Repetição/Ciclo** - Normalmente utilizadas quando se deseja executar uma sequência de instruções repetidas vezes. A estrutura de repetição, tal como a de seleção, envolve sempre a avaliação de uma condição. A nossa linguagem suporta quatro tipos de estruturas de repetição: com teste no início (**enquanto - faça**), com teste no fim (**faça - enquanto; repita - ate**) e com variável de controlo (**para - passo**).

   - **Repetição com teste no início**

     Sintaxe:

     enquanto (<condição>) faca

comando

fimenquanto

- Enquanto a condição for verdadeira o comando é executado até que a condição seja falsa. Por Exemplo:

```
enquanto (x >2) faca
        escreva "x ainda é menor que 2"
        x ← x - 1
fimenquanto
```

Exemplo 4.15: Exemplo de uma Estrutura de Repetição "ENQUANTO".

- **Repetição com variável de controlo**

Sintaxe:

para <variável>de <valor1>ate <valor2>passo <valor3>

comando

fimpara

- Para a <variável>desde o <valor1>até <valor2>o comando será executado até que <valor1 >seja maior que <valor3>, a palavra chave **passo** significa que o <valor3>será incrementado/decrementado na <variável>. Por exemplo:

```
para x de 10 ate 2 passo -1
        escreva "x ainda é menor que 2"
fimpara
```

Exemplo 4.16: Exemplo de uma Estrutura de Repetição "PARA".

- **Repetição com teste no fim**

Sintaxe 1:

faça

comando

enquanto (condição)

- Repete um conjunto de instruções até que a condição seja falsa, ou seja, as instruções são executadas sempre que a condição for verdadeira. Por exemplo:

```
faca
        escreva "x ainda é menor que 2"
        x ← x - 1
enquanto (x >2)
```

Exemplo 4.17: Exemplo de uma Estrutura de Repetição "FAÇA".

Sintaxe 2:
repita
    comando
ate (condição)

- Repete um conjunto de instruções até que a condição seja verdadeira, ou seja, as instruções são executadas sempre que a condição for falsa. Por exemplo:

```
repita
        escreva "x ainda é menor que 2"
        x ← x - 1
ate (x = 2)
```

Exemplo 4.18: Exemplo de uma Estrutura de Repetição "REPITA".

4.4.1.11 Métodos (Funções e Procedimentos)

São sub-programas que são criados antes da instrução de **início** do programa. Podem ser funções (métodos que retornam valor de um tipo específico) ou procedimentos.

- **Procedimento**

  Sintaxe:

  procedimento <nome_procedimento>(<parâmetros>)

        comandos

  fimprocedimento

```
procedimento avaliador (inteiro x)
    repita
        escreva "x ainda é menor que 2"
        x ← x - 1
    ate (x = 2)
fimprocedimento
```

Exemplo 4.19: Exemplo de um Procedimento.

Por exemplo:

- **Função**

  Sintaxe:

  funcao <tipo de retorno ><nome_função >(<parâmetros >)

  comandos

  retorne <valor do tipo de retorno>
  fimfuncao

  Por exemplo:

```
funcao inteiro avaliador (inteiro x)
    repita
        escreva "x ainda é menor que 2"
        x ← x - 1
    ate (x = 2)
    retorne x
fimfuncao
```

Exemplo 4.20: Exemplo de uma Função.

4.4.1.12 Métodos Pré-Definidos

Nesta Secção apresentamos todos os métodos existentes na nossa linguagem, criadas com intuito de facilitar algumas operações durante a resolusão de alguns problemas. Existem três grupos de métodos: `sem argumento`; `com um argumento`; e os de `dois argumentos`. Estão classificados em `funções matemáticas`, `funções de texto` e `especiais`, tais como: `LIMPATELA` e `PARAR`. A Tabela 4.6 apresenta todos os métodos e a descrição dos mesmos.

Método	Descrição
LIMPATELA	Permite limpar a consola
PARAR	Permite parar a execução a meio do código
MENOR (arg1, arg2)	Retorna o menor de dois valores reais
MAIOR (arg1, arg2)	Retorna o maior entre dois valores reais
POTENCIA (arg1, arg2)	Potenciação de dois números reais
COMPRIMENTO (argumento)	Retorna a quantidade de caracteres de um texto
ARREDONDAR (argumento)	Arredondamento um número real
PARTEFRAC (argumento)	Parte fracionaria de um número real
PARTEINTEIRA (argumento)	Parte inteira de um número real
LN (argumento)	Retorna o Logaritmo Natural
LOG (argumento)	Retorna o Logaritmo base de 10
RAIZ (argumento)	Retorna a Raiz quadrada
ABS (argumento)	Retorna o Valor absoluto de um número real
EXP (argumento)	Calcula a Exponencial de um número real
SEN (argumento)	Calcula o Seno de "argumento"
COS (argumento)	Calcula o Co-seno de "argumento"
TAN (argumento)	Calcula a Tangente de "argumento"
CTG (argumento)	Calcula o Co-tangente de "argumento"
ASEN (argumento)	Calcula o Arco Seno de "argumento"
ACOS (argumento)	Calcula o Arco Co-seno de "argumento"
ATAN (argumento)	Calcula o Arco Tangente de "argumento"
ACTG (argumento)	Calcula o Arco Co-tangente de "argumento"
SENH (argumento)	Calcula o Seno hiperbólico de "argumento"
COSH (argumento)	Calcula o Co-seno hiperbólico de "argumento"
TANH (argumento)	Calcula o Tangente hiperbólica de "argumento"
CTGH (argumento)	Calcula o Co-tangente hiperbólica de "argumento"

Tabela 4.6: Métodos Pré-Definidos

4.4.1.13 Especificação *Backus Normal Form* (BNF)

Nesta Secção nos dedicamos a apresentar a especificação BNF da nossa pseudo-linguagem, que é uma meta-sintaxe usada para expressar gramáticas livres de contexto, ou seja um modo formal de descrever linguagens formais. Entretanto segue abaixo as regras gramaticais da nossa linguagem:

S = “inicio” novalinha comandos “fimAlgoritmo” novalinha

digito = [“0”| ”1”| ”2”| “3”| ”4”| ”5”| ”6”| ”7”| “8”| ”9”]

letras-minusculas = [“a”|”b”|”c”|”d”| “e”| “f”| “g”| “h”| “i”| “j”| “k”| “l”| “m”| “n”| “o”| “p”|“q”| “r”|”s”|”t”| “u”| “v”| “w”| “x”| “y”| “z”| “á”| “é”| “í”| “ó”| “ú”| “ý”|“ã”| “õ”|“ñ”| “à”| “è”| “ì”| “ò”| “ù”| “ä”| “ë”| “ï”| “ö”| “ü”| “ÿ”| “â”| “ê”|“î”| “ô”| “û”| “ç”]

letras-maiusculas = [“A”|”B”|”C”|”D”| “E”| “F”| “G”| “H”| “I”| “J”| “K”| “L”| “M”| “N”| “O”|“P”| “Q”| “R”|”S”|”T”| “U”| “V”| “W”| “X”| “Y”| “Z”| “Á”| “É”| “Í”| “Ó”|“Ú”| “Ý”| “Ã”| “Õ”| “Ñ”| “À”| “È”| “Ì”| “Ò”| “Ù”| “Ä”| “Ë”| “Ï”| “Ö”|“Ü”| “Ÿ”| “Â”| “Ê”| “Î”| “Ô”| “Û”| “Ç”]

letra = (letras-minusculas|letras-maiusculas)

numérico = digito { digito}

numero = [+|-] numérico

valor = numero | identificador

literal = "{*}"

logico = “verdadeiro” | ”falso”

identificador = letraletra | digito | ”_”

novalinha = "\n"

operador-logico = “não” | “e” | “ou”

operador-numerico = “+” | “-“ | “*” | “/”

operador-relacional = “<” | “ >” | “ >=” | “<=” | “=” | “<>”

comandos = (declaração | procedimento | est-repetição | est-cond | atribuição | funcao | assinatura) comandos

identificadores = identificador “,” identificador

declaração = (tipo|registo|matriz) identificadores novalinha

tipo = “lógico” | “literal” | "caracter" | "inteiro" | "real"

registo = “registo” identificador novalinha (tipo | registo | matriz) identificadores novalinha (tipo | registo | matriz) identificadores novalinha “fimregisto” novalinha

matriz = tipo identificador [valor | {“,” valor}]

procedimento = “procedimento” identificador ( ) | parâmetro novalinha comandos “fimprocedimento” novalinha

parâmetro = (tipo identificador {identificadores}) novalinha

funcao = “função” tipo identificador parâmetro novalinha comandos “fimfunção” novalinha

assinatura = "procedimento" | "funcao" (tipo {"," tipo}) novalinha

estrutura-condicional = “se” expressao-logica “então” novalinha comandos novalinha [“senão” novalinha comandos novalinha] “fimse” novalinha | “escolha” identificador novalinha “caso” expressao “:” novalinha comandos “caso” expressao “:” novalinha comandos | “defeito” expressao “:” novalinha comandos “fimescolha” novalinha

estrutura-repeticao = “para” identificador “de” valor “ate” valor “passo” valor novalinha comandos “fimpara” novalinha | ”repita” novalinha comandos “ate” expressao-logica | ”faca” novalinha comandos “enquanto” expressao-logica | ”enquanto” expressao-logica novalinha comandos “fimenquanto”

atribuicao = identificador "←" expressao novalinha

expressao = expressao-logica | expressao-literal | expressao-numerica

expressao-numerica = valor operador-numerico expressão-numerica | [“+”|”-“] valor | “(“expressão-numerica”)”

valor-logico = logico | identificador

expressao-logica = valor-logico | expressão-numerica operador-relacional expressao-numerica | valor-logico operador-logico expressao-logica

4.4.2 Exemplos de Programas

Reservamos esta Secção para apresentar alguns exemplos completos e mais práticos de programas desenvolvidos na nossa ferramenta. A ideia consiste em passar uma noção mais clara da sintaxe do CompAlg.

Para todas as figuras apresentadas abaixo: 1) é a área de código do compAlg e 2) é a consola de resultados do programa.

- **1º Exemplo**: Este primeiro exemplo consiste em apresentar um pequeno programa que dado 10 valores inteiros, determina e apresenta o maior valor entre eles. Abaixo apresentamos a ilustração do programa na nossa ferramenta.

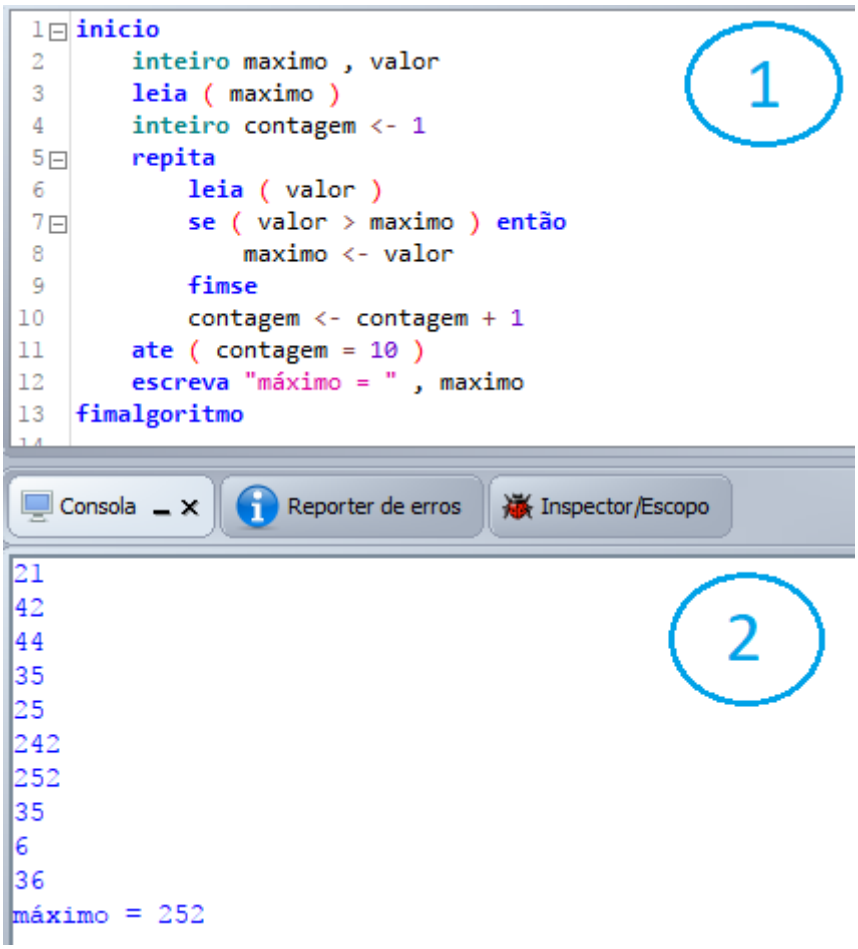


Figura 4.9: Programa que Determina o Maior de 10 Valores Inseridos pelo Teclado

No Exemplo 4.9, exploramos algumas instruções úteis da nossa linguagem como: a declaração de variável, entrada e saída de dados e estrutura de repetição com

teste no fim. Podemos observar na consola a entrada de dez valores inteiros e o resultado é o maior valor entre eles.

- **2º Exemplo**: O segundo exemplo consiste em apresentar um pequeno programa que dado um vetor de 10 posições com valores inteiros, conta e apresenta o número de ocorrências de um determinado valor.

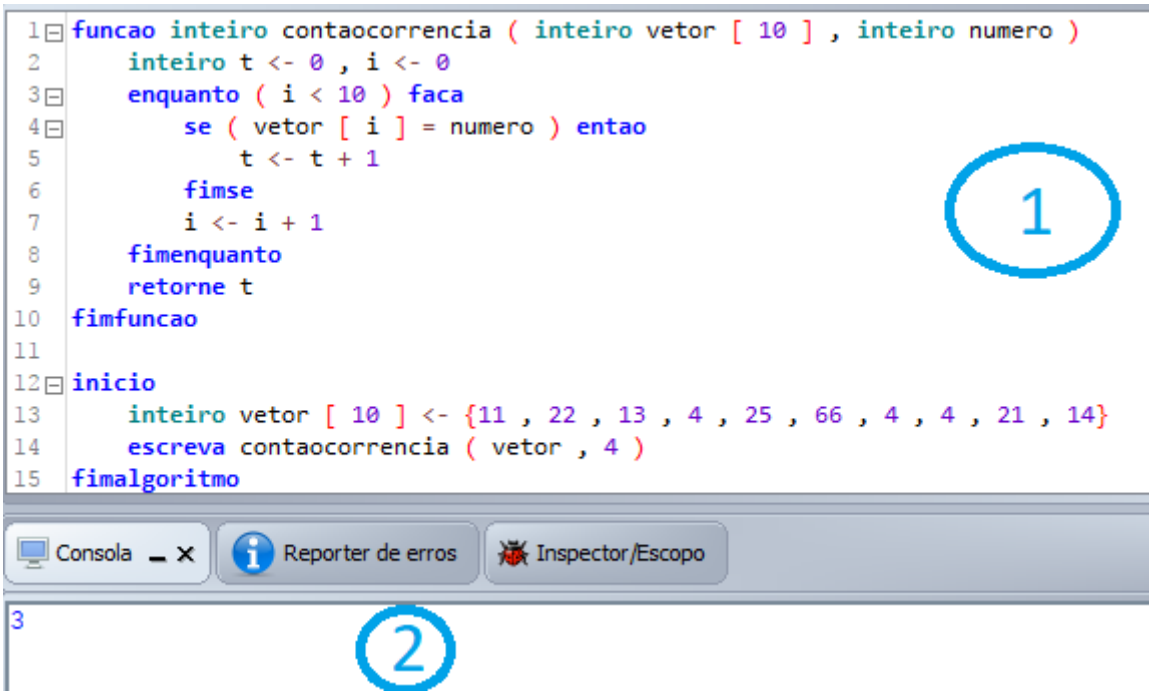


Figura 4.10: Programa que Conta Ocorrência de um Valor num Vetor de Inteiros

No Exemplo 4.10, exploramos a utilização de função e vetor. Temos um vetor inicializado com dez elementos (linha 13) e em seguida invocamos a função "contaocorrencia" (linha 14) que recebe como argumento o vetor (passagem por refência) e o valor 4 (o valor que queremos saber a ocorrência no vetor).

- **3º Exemplo**: O Exemplo 4.11 é o clássico algoritmo de ordenação por seleção, que dado um vetor com 10 posições de inteiros, determina e apresenta o vetor em ordem crescente.

  Ainda sobre o nosso clássico exemplo de ordenação por seleção, na linha 17 declaramos e inicializamos o nosso vetor com dez valores inteiros e distintos, na linha 18 invocamos o nosso procedimento "selecao" responsável pela ordenação, passando como argumento o vetor (passagem por referência) e o número de elementos a ordenar. Em seguida (na linha 19), imprimimos o vetor.

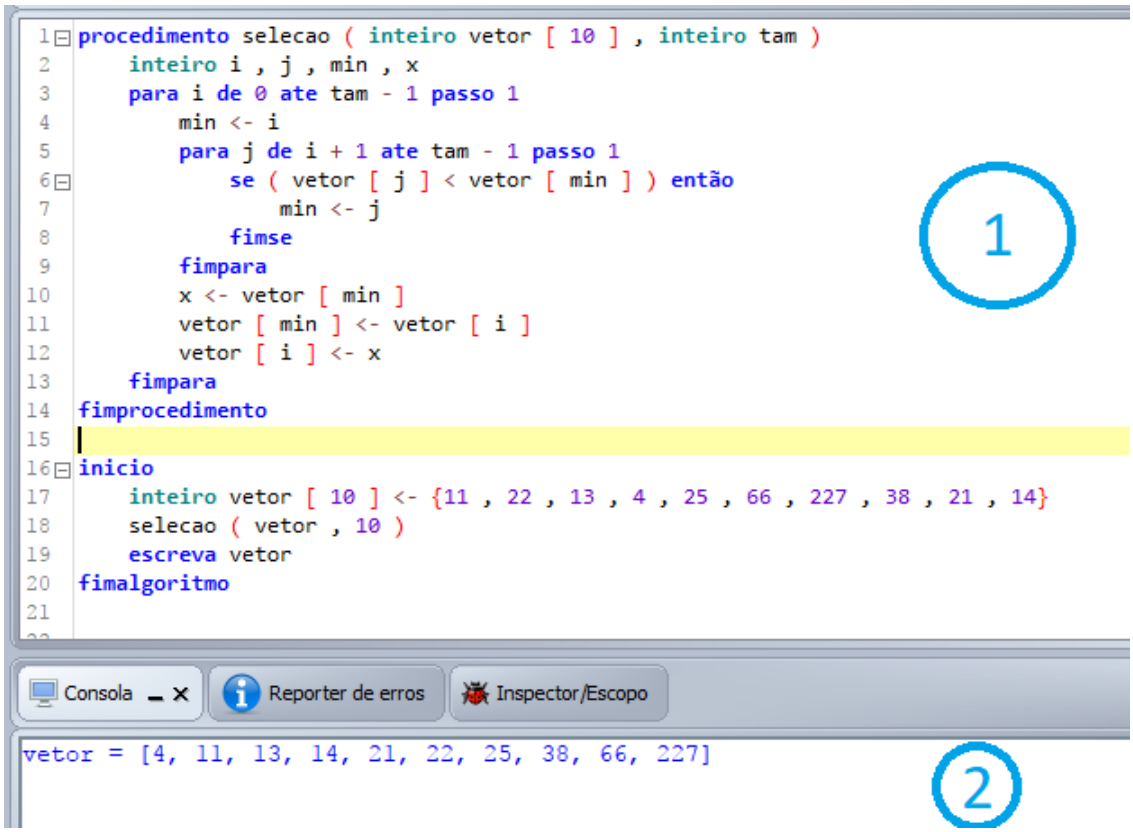


Figura 4.11: Programa que Ordena um Vetor de Inteiros por Seleção

- **4º Exemplo**: Pretendemos com o exemplo na Figura 4.12 apresentar a utilização da estrutura de dados do tipo **registo**. O programa consiste em cadastrar de uma pessoa.

  Definimos dois **registos** (cadastro e dados) nas linhas 1-10, na linha 13 definimos um vetor com duas dimensões, do tipo do registo "dados" criado na linha 6. Em seguida cadastramos os dados das pessoas nas linhas 14-24, posteriormente apresentamos o nome (na linha 28) ou a idade (na linha 30) mediante uma condição (linha 27).

- **5º Exemplo**: Neste último exemplo vamos mostrar um pequeno programa que utiliza **assinaturas**. O programa consiste em simular chamadas sucessivas de funções (**primeiro** e **segundo**) que chamam-se mutuamente sem sua definição estar completa.

  Na Figura 4.13, exploramos a utilização de assinaturas para permitir chamadas mútua das funções (**primeiro e segundo**). Nas linhas 1 e 2 assinamos as funções, na linha 21 imprimimos o valor de retorno da função **primeiro** que recebe como argumento uma variável inteira com valor 20. Esta função retorna

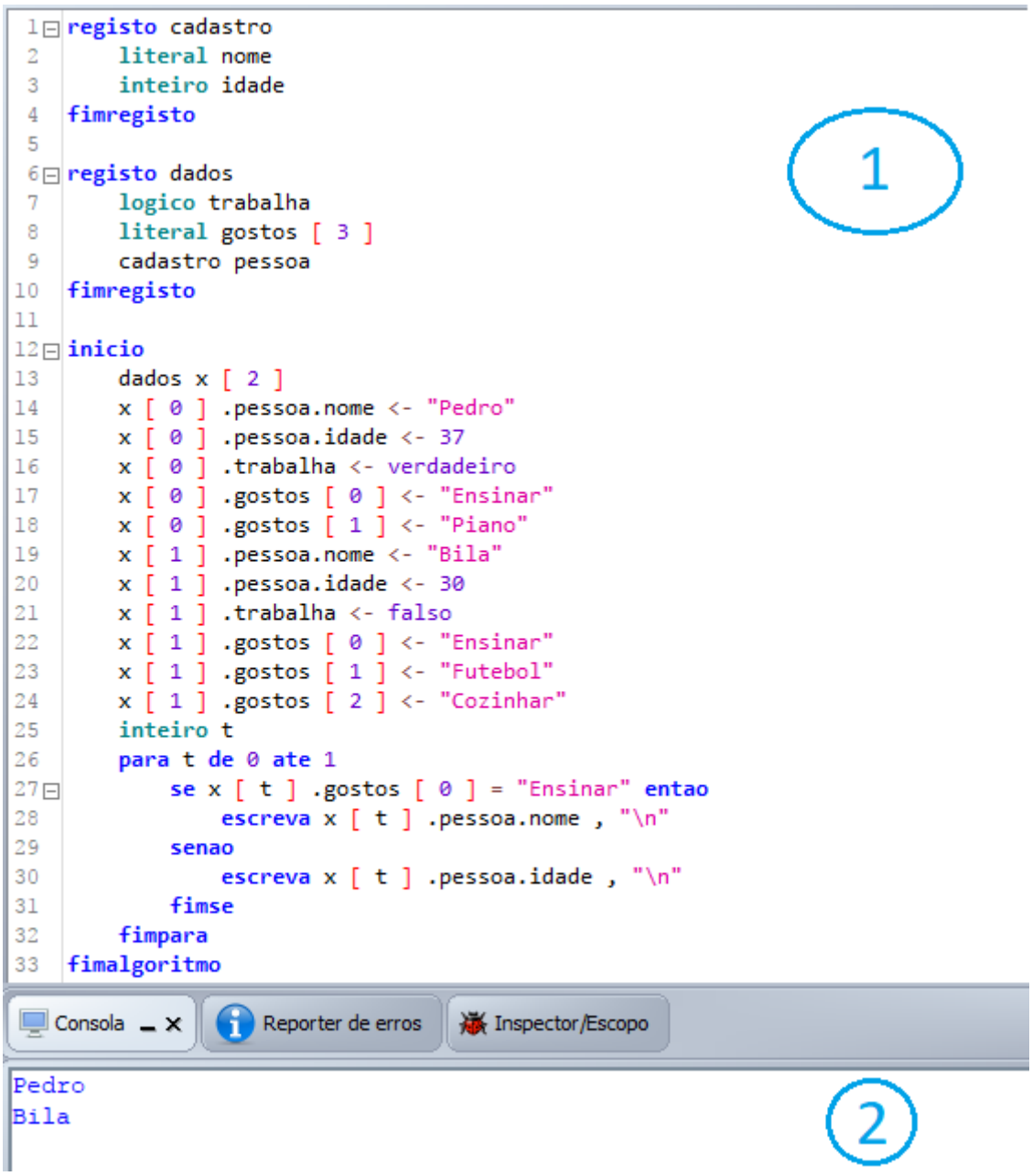


Figura 4.12: Programa que Cadastra os Dados de uma Pessoa

1, caso o argumento passado seja igual a 0 (zero) ou invoca a função **segundo** com menos 1 no valor do caso do argumento caso contrário, que por sua vez retorna 0 (zero) se o argumento for zero ou chama novamente a função **primeiro** com menos 1 no argumento, assim sucessivamente até que o valor do argumento seja zero em qualquer uma das funções.

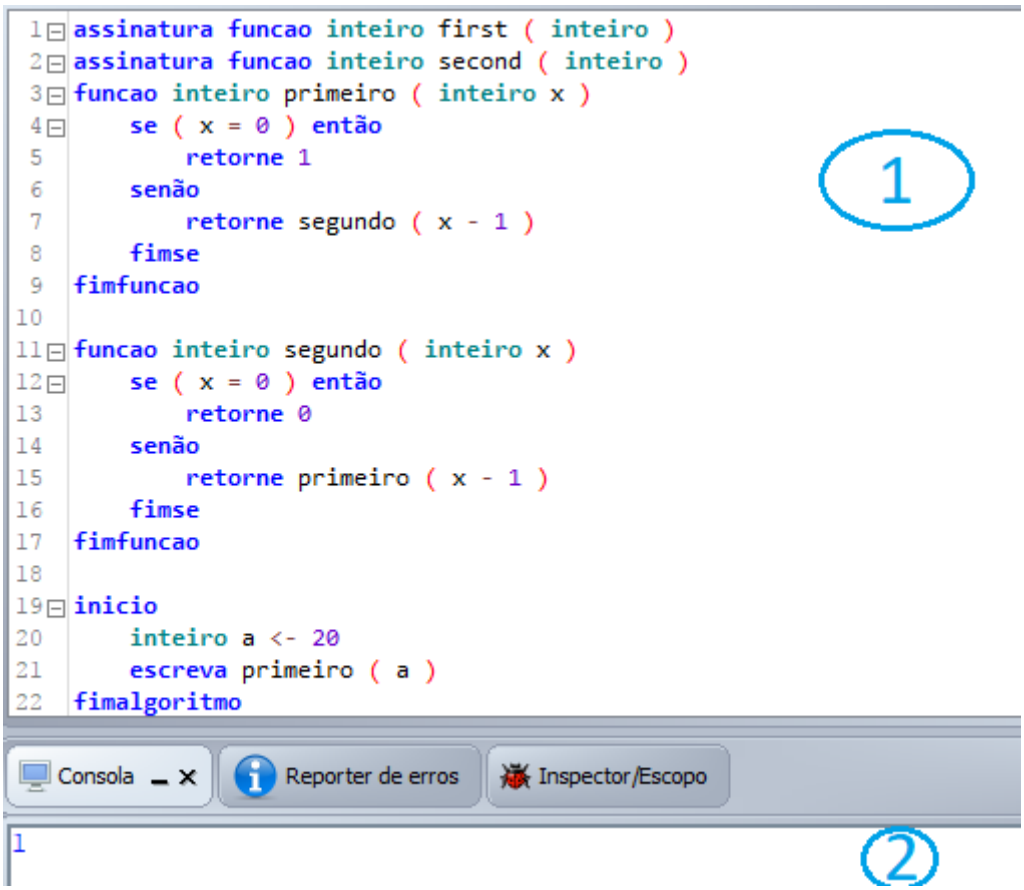


Figura 4.13: Programa que Simula Chamada Mútua de Funções

4.5 Síntese

Neste capítulo começamos por apresentar um breve histórico da nossa ferramenta e suas versões e problemas encontrados. Depois mostramos uma visão geral do estado atual da ferramenta (Secção 4.2) mas sem entrar em detalhes do seu funcionamento que está reservado no Capítulo 6. Posteriormente seguiu-se a abordagem da metodologia utilizada (Secção 4.3), onde falamos sobre algumas questões que envolve a engenharia de *software*. Depois falamos com mais propriedade sobre a nossa linguagem de programação, onde vimos os detalhes da linguagem (Secção 4.4) e o fluxo de informação da nossa ferramenta (Secção 4.3.3). E terminamos este capítulo com cinco exemplos de programas construídos na nossa ferramenta com intuito de mostrar a sintaxe da nossa linguagem (Secção 4.4.2).

O Processo de Compilação 5

Neste capítulo, pretendemos abordar sobre o processo de compilação da nossa ferramenta desde a entrada do código fonte até a geração do código alvo. Apresentamos as partes envolvidas no processo e as tarefas de cada uma dessas partes. Mas inicialmente vamos conceituar sobre compilador e depois compreender as partes.

Um `compilador` é um programa de computador que traduz um programa-fonte, geralmente escrito em linguagem de alto nível, para uma outra linguagem de baixo nível, a linguagem compreendida pela máquina/computador [Hol90]. Consideremos a figura abaixo:

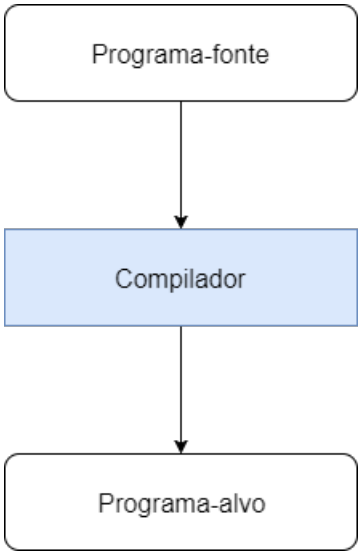


Figura 5.1: Esquema Simplificado de Compilação.

O esquema da Figura (5.1) mostra de uma forma bem simples o fluxo de entrada e saída de um mesmo código em um compilador, como entrada temos um programa-fonte que ao passar pelo compilador é convertido em um programa-alvo. Um programa-fonte pode ser escrito por exemplo em Java, C#, C++, Python entre outras linguagens, já o programa-alvo que é a linguagem que a máquina vai compreender é gerada normalmente para `Linguagem Assembly` ou `Bytecodes`.

No caso do nosso compilador o programa-fonte é escrito na nossa linguagem descrita na Secção 4.4 e é convertido numa linguagem intermediária que é o Java pelo compilador. Esse "novo" código é então compilado pelo compilador do Java, o `Javac`, que por sua vez converte o programa gerado em Java para `Bytecodes` que é enviado para o processamento do computador através do `Java Virtual Machine (JVM)`. Logo após todo o processamento realizado, a máquina retorna o resultado ou resposta do código para o compilador Java que o envia para o CompAlg, e este retorna com a resposta na consola. Para melhor compreensão, abaixo mostramos um esquema ilustrando o fluxo de informações entre os compiladores.

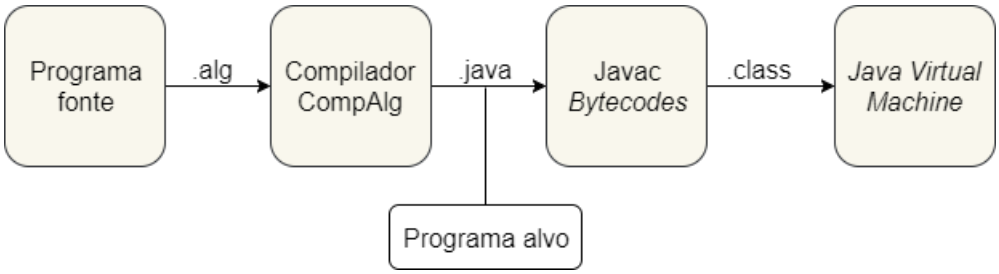


Figura 5.2: Esquema Simplificado do Processo de Compilação no CompAlg.

Para compreender melhor o funcionamento do compilador da nossa ferramaneta fazer uma descrição mais detalhada da sua estrutura. O CompAlg, assim como outros compiladores [Hol90], é constituído de quatro etapas que estão ilustradas na figura 5.3. A **primeira etapa** trata-se de um pré-processador que lê o código fonte e identifica os caracteres das unidades significativas denominadas de "marcas". A **segunda etapa**, que é considerada como o coração do compilador, é composta por quatro elementos: `Analisador léxico (AL), Analisador, Tabela de símbolos e Gerador de códigos`. Enquanto o analisador léxico é responsável por organizar as marcas, o analisador trata da análise sintática do programa e tem como resultado uma árvore de análise sintática ou árvore sintática. Já a tabela de símbolos é uma estrutura de dados, utilizada para o armazenamento de informações de identificadores, tais como constantes, funções, variáveis e tipos de dados. E, o último elemento da segunda etapa é o gerador de código que gera o código alvo do compilador em questão. A **terceira etapa** é responsável por uma otimização do código fonte que agora vai ser convertido para a nossa linguagem intermediária, após a análise semântica, para isso a tabela de símbolos é visitada várias vezes até que o código seja traduzido de forma eficiente. E por sua vez, a **quarta etapa** trata da obtenção do código alvo.

Nas secções seguintes vamos detalhar um pouco o funcionamento dessas partes no nosso compilador, não exatamente como vimos no esquema proposto por [Hol90], como

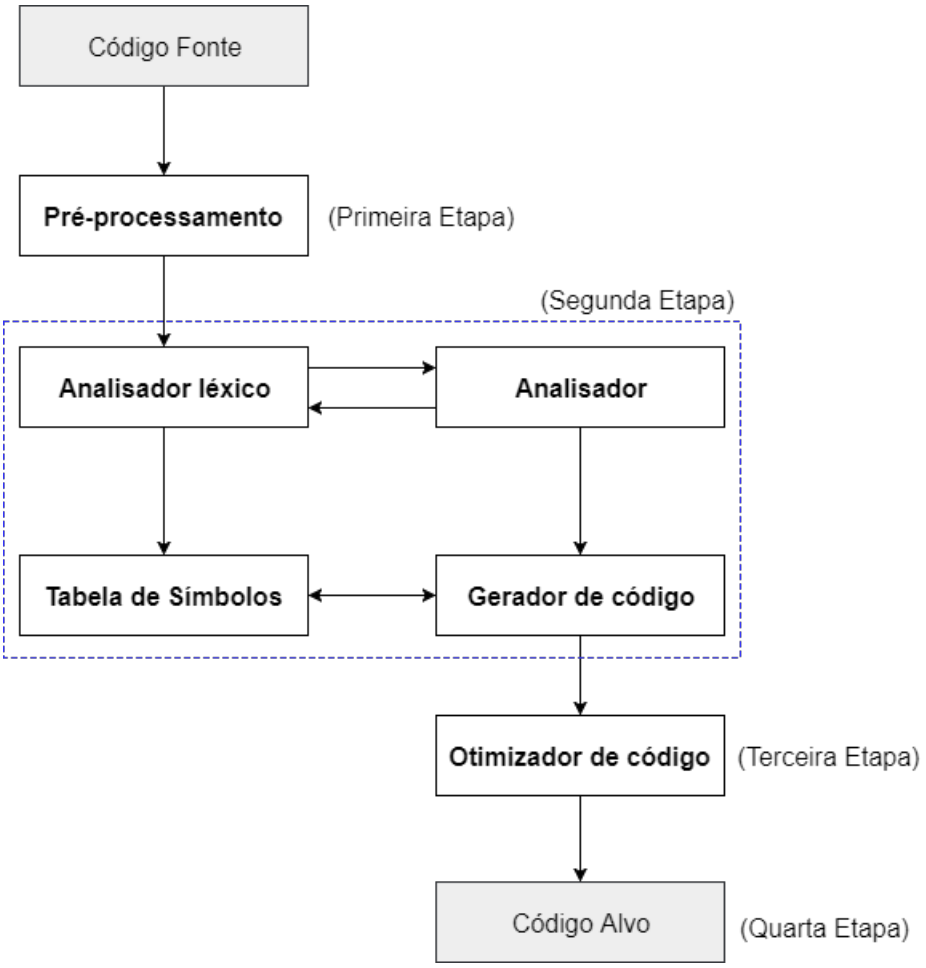


Figura 5.3: Etapas de um Compilador (Holub [Hol90] pag.2, Fig. 1.1)

mostra a Figura (5.3) mas que segue fielmente a ideia dessas etapas.

5.1 Analisador Léxico (AL)

O analisador léxico faz uma leitura do código fonte que reconhece os símbolos distinguindo aqueles que pertencem a linguagem e informando quando algum símbolo não faz parte dela. É interessante saber que durante essa "varredura" o AL não se preocupa em verificar se os símbolos estão na ordem correta de uso no código. A base do nosso AL foi construída com ajuda da utilização da ferramenta **JFlex**, que é um gerador de analisadores léxicos totalmente em Java [Dé98]. Esta ferramenta possui uma sintaxe de especificações inspiradas na sintaxe das especificações para **Lex**, que é o gerador de analisadores léxicos para *Unix*.

Na Figura 5.4 temos uma ilustração do fluxo de informação do analisador léxico

no compilador, podemos verificar que o programa fonte entra no analisador léxico, tendo em subsequência uma troca de informações com a tabela de símbolos e o parser (analisador), fazendo assim a identificação dos símbolos (lexemas).

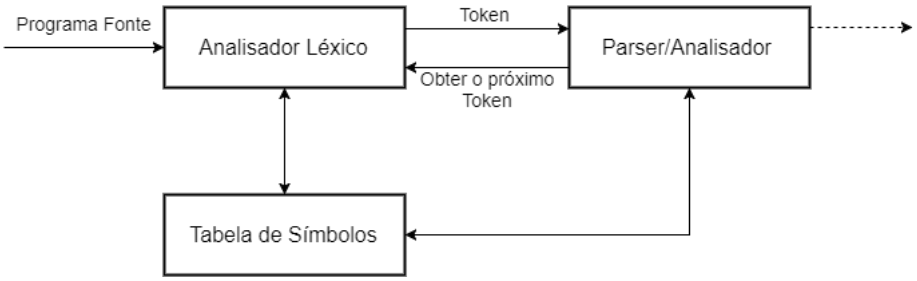


Figura 5.4: Fluxo de Informações no Analisador Léxico

Para deixar um pouco mais clara a ideia, temos abaixo uma tabela com uma pequena parte de um programa fonte. Uma avaliação condicional para um certo valor que pode ser menor que 10, "Se media <10 então". Os lexemas (símbolos) deste programa estão listados na segunda coluna da tabela, enquanto os `tokens` estão na terceira coluna. Na quarta coluna está a descrição informal de cada um desses símbolos que, como podemos ver, podem ser identificadores, palavras reservadas, constantes inteiras, entres outros.

Programa Fonte	Lexema	Token	Descrição Informal
Se media <10 entao	Se	strSE	Palavra reservada
	media	strINDENTIFICADOR	Identificador
	<	MENOR_QUE	Operador relacional
	10	INT_LITERAL	Constante inteira
	entao	ENTAO	Palavra reservada

Tabela 5.1: Esquema de Programa-Fonte, Lexemas e Tokens.

O analisador léxico também é utilizado para realizar outras funções secundárias como remover espaços em branco ou comentários e gerar mensagens de erro referindo-se a linhas no código.

5.2 Analisador Sintático (AS)

O analisador sintático lê o código fonte e identifica se os símbolos estão dispostos na ordem correta para nossa linguagem. A gramática do `AS` é livre de contexto, e é nessa gramática que constam todas as regras que são válidas para a linguagem em questão, como expressões lógicas, comandos e declarações de variáveis. Tais regras estão definidas na Secção 4.4.1.13 (especificações BNF). O `AS` constrói uma árvore sintática que descreve as construções da linguagem nas sequências lógicas que foram

reconhecidas no programa. O `AS` também identifica os erros sintáticos. Um exemplo de árvore sintática pode ser visualizado na Figura 5.5, que é a árvore sintática para a expressão simples "X ← 9 - 7".

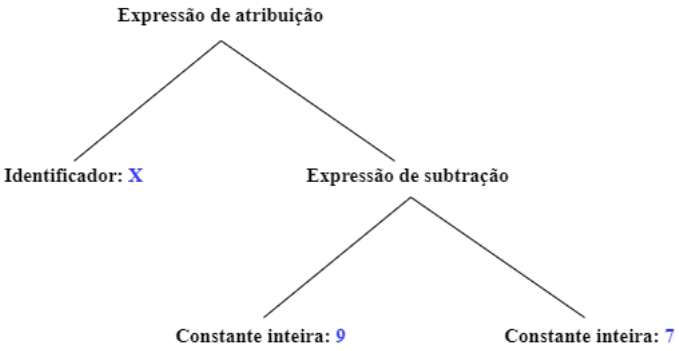


Figura 5.5: Exemplo de Árvore Sintática

5.3 Analisador Semântico (ASem)

Por sua vez, o analisador semântico faz uma busca por coerência nas construções utilizadas pelo programador. O `ASem` não faz sua análise no código em si, ele examina a árvore sintática e busca por sequência compatível na tabela de símbolos, que contém informações sobre funções, declarações de variáveis, tipos estruturados entre outros. Algumas das incoerências que o analisador semântico procura são do tipo:

- Uso de constantes, funções ou variáveis não declaradas;
- Solicitações de funções com tipos ou quantidades incorretas;
- Declaração duplicada de variáveis, funções ou constantes;
- Uso de operadores incompatíveis com os operandos, como por exemplo uso do operador "*" (multiplicador) entre duas variáveis *string* (cadeias de caracteres).

É interessante salientar que os erros acima citados não são identificados pelo analisador sintático, pois não desobedecem as regras gramaticais da nossa linguagem (ver a Secção 4.4.1.13), e assim só podem ser capturados pelo `ASem`.

5.4 Gerador de Código

Quando não há erros sintáticos ou semânticos o gerador de código cria o código na linguagem alvo, que é a linguagem da máquina ou uma linguagem intermediária dependendo do compilador. No caso do CompAlg o gerador de código cria um código intermédio em Java para ser compilado pelo Java (Javac). Esse programa gerado reflete as instruções de baixo nível do código escrito no programa fonte. Em compiladores genéricos esta etapa de geração de código, costuma ser bastante complexa por depender além da linguagem também da arquitetura e do sistema operativo do computador, mas no caso da nossa ferramenta não se torna tão complexa, pois depende da máquina virtual do Java (JVM), que assumimos estar bem arquitetada.

5.5 Síntese

Abordamos sobre todo o processo de compilação da nossa ferramenta. Começamos por definir um compilador genérico e as partes fundamentais que o constituem (ver Secção 5.3), depois falamos de cada parte e como funciona o nosso compilador, destacamos: O Analisador Léxico, Sintático, Semântico e o Gerador de Código (Secções 5.1, 5.2, 5.3 e 5.4). E posteriormente afirmamos que nosso compilador transforma o código fonte (escrito na nossa linguagem) para a linguagem Java (linguagem intermédia), que posteriormente é compilado pelo Javac e nos retorna os resultados após ter sido processado pela máquina virtual do Java (ver Secção 5.2).

Funcionalidades da Ferramenta 6

Neste capítulo dedicamos-nos a apresentar de forma detalhada as funcionalidades existentes na nossa ferramenta. Numa primeira fase vamos mostrar as funcionalidades mais básicas e posteriormente mostramos aquelas mais complexas.

6.1 Edição, Depuração e Execução de um Programa

Foram já apresentadas no Capitulo 4, mais precisamente na Secção 4.2, as partes que constituem a nossa ferramenta. Nesta Secção seremos mais precisos e focados nas tarefas de cada uma das partes.

Para edição, depuração e execução de um programa mostraremos de forma gráfica as partes que estão envolvidos nesse processo. São eles: Editor de código; botão de depuração; botão de execução; consola; repórter de erros. As Figuras 6.1,6.2 e 6.3 mostram o funcionamento das partes citadas.

- **Editor de Código**

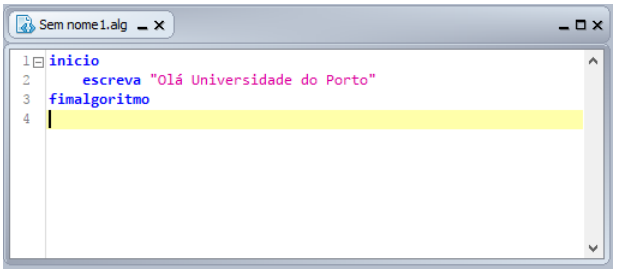


Figura 6.1: Funcionamento do Editor

As Figuras 6.1 e 6.1 ilustram as funcionalidades do nosso editor, totalmente moderno, dentre as várias funcionalidades destacamos:

- Apresentação dos números de linhas do código.
- Realça a linha que contém erros.

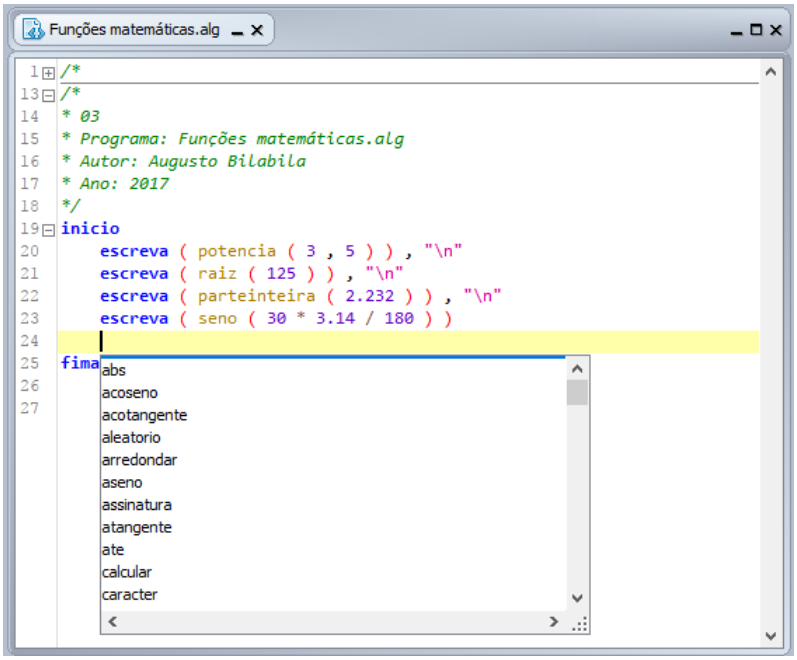


Figura 6.2: Funcionamento do Editor com Fecho e Ativação do Auto-Completar

- Ativa o fecho quando necessário em blocos.
- Ativa o auto-completar com das palavras-chaves quando pressiona a combinação das teclas (`CTRL + SPACE`).

- **Ação de Depuração e de Execução**

Figura 6.3: Depuração e Execução

A Figura 6.3 mostra os botões para depuração (a esquerda) e o botão de execução (a direita). O depurador consiste em acionar o compilador para a verificação de erros providos do código fonte (Editor) e envia uma informação ao repórter de erros (Figura 6.6), que informa o programador sobre o resultado da depuração. O botão de execução é mais completo, pois para além de executar o programa colocando os resultados na consola, também faz a depuração do programa.

- **Consola**

A Figura 6.4 mostra a nossa consola, onde são apresentados os resultados do código fonte compilado. A Figura 6.5 mostra também a entrada de valor pelo teclado através da consola.

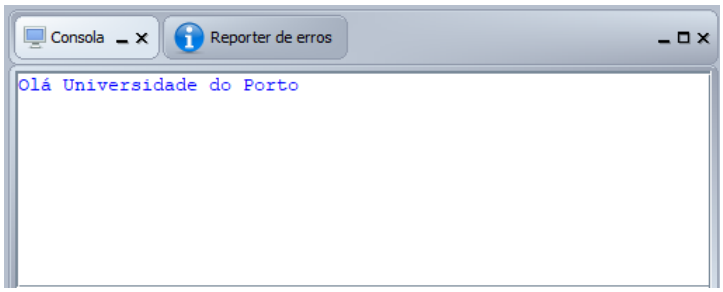


Figura 6.4: Consola da Nossa Ferramenta

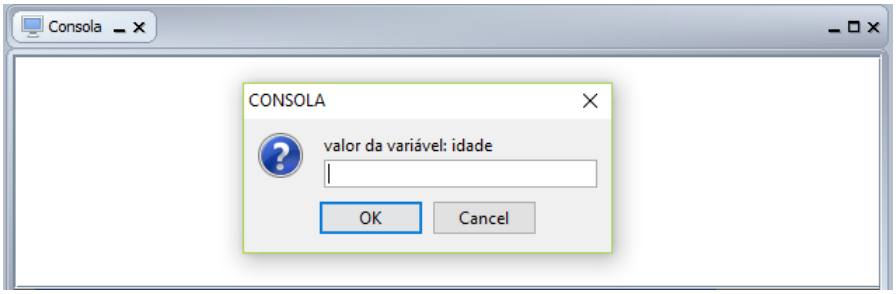


Figura 6.5: Recebendo Valor pelo Teclado via Consola.

- **Repórter de Erros** O repórter de erros tem como função informar ao programador sobre o estado do seu programa, isto é, de erros de compilação ou de execução como mostra a Figura 6.7.

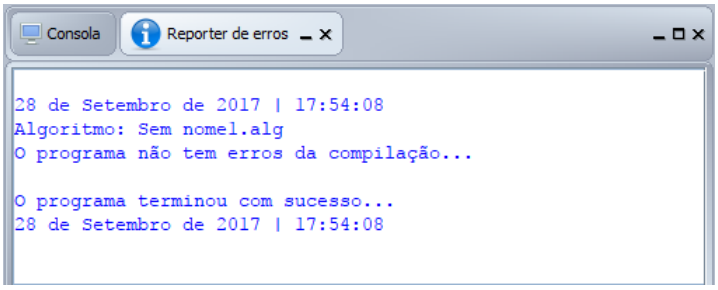


Figura 6.6: Reportando Sobre o Estado da Depuração e Execução.

6.2 Impressão de uma Estrutura de Dados

Um dos grandes desafios desta ferramenta é o tratamentos das estruturas de dados. Nesta secção vamos apresentar de forma simples como podemos imprimir uma estrutura. A Figura 6.8 ilustra essa funcionalidade do compAlg ao nível da linguagem.

Vimos de forma clara que podemos imprimir um vetor que contém uma estrutura complexa que é o registo. Sabemos que ainda existem algumas limitações, no entanto

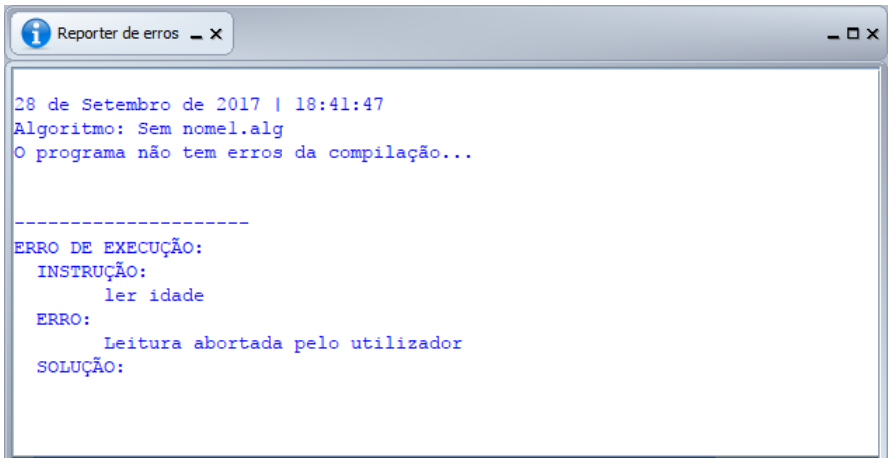


Figura 6.7: Reportando Erro na Execução.

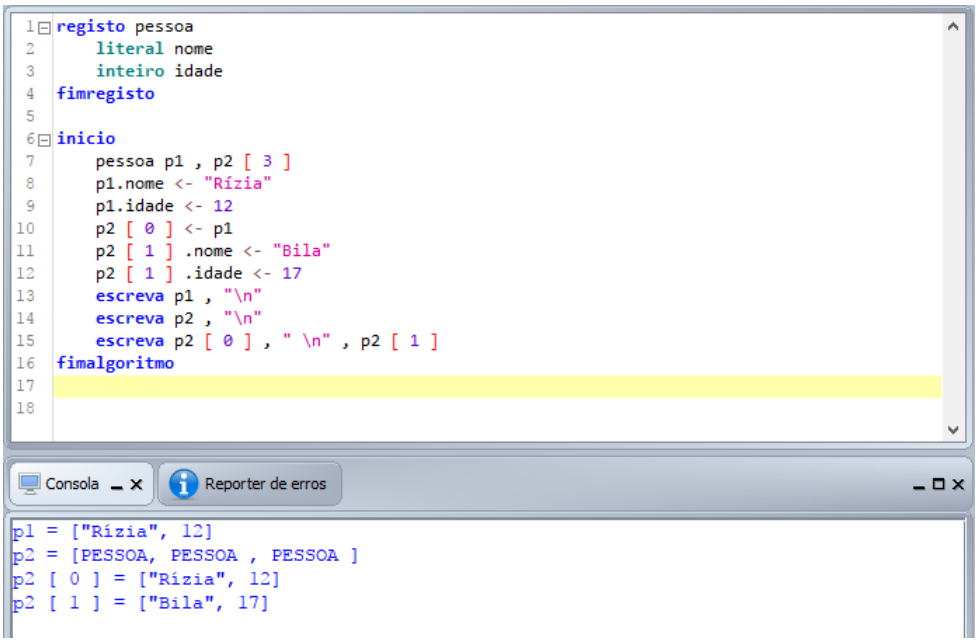


Figura 6.8: Imprimir uma Estrutura de Dados

essa funcionalidade é útil para o programador noviço, que poderá dar um passo maior na sua aprendizagem.

Importante realçar que nessa funcionalidade estamos interessados em permitir que o conteúdo de uma estrutura complexa seja apresentada para fins de estudos.

6.3 Execução Passo a Passo

A execução passo a passo é sem dúvida uma das grandes funcionalidades da nossa ferramenta permitindo que, visualizemos o **conteúdo das variáveis durante a execução** de cada linha do programa fonte, dando também a possibilidade de **inspecionar de forma individual o conteúdo dos vetores/matrizes e dos registo**. Com a implementação da execução passo a passo também temos a possibilidade de **averiguar o valor booleano de cada avaliação condicional** do programa fonte e ainda incluindo a funcionalidade que permite **parar a execução a meio do programa** (utilizando a instrução **parar** no código fonte). Nas Figuras 6.9 e 6.10, mostramos como as funcionalidades referidas nesta secção são executadas e apresentadas graficamente.

- **Visualizar o Conteúdo das Variáveis**

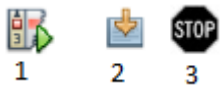


Figura 6.9: Ativar e Executar Passo a Passo

A Figura 6.9 mostra a ativação da execução passo a passo, o passo 3 (*stop*) é opcional, ou seja é útil quando se deseja parar a execução passo a passo.

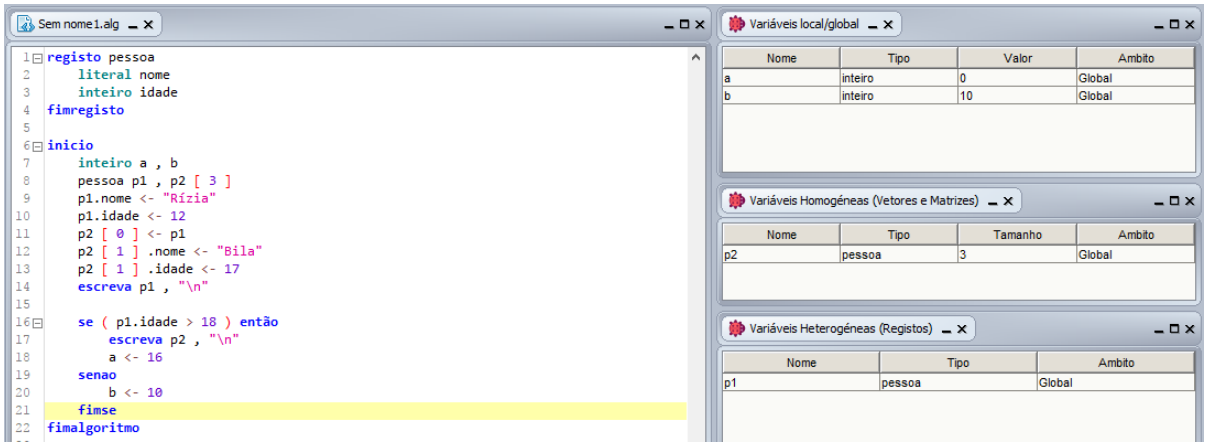


Figura 6.10: Estado das Variáveis Durante a Execução Passo a Passo

Na Figura 6.10 mostramos (à esquerda) um pequeno programa que posteriormente foi executado passo a passo. O visualizador das variáveis é apresentado à direita da Figura 6.10, onde aparecem as variáveis simples, vetores/matrizes e os registos.

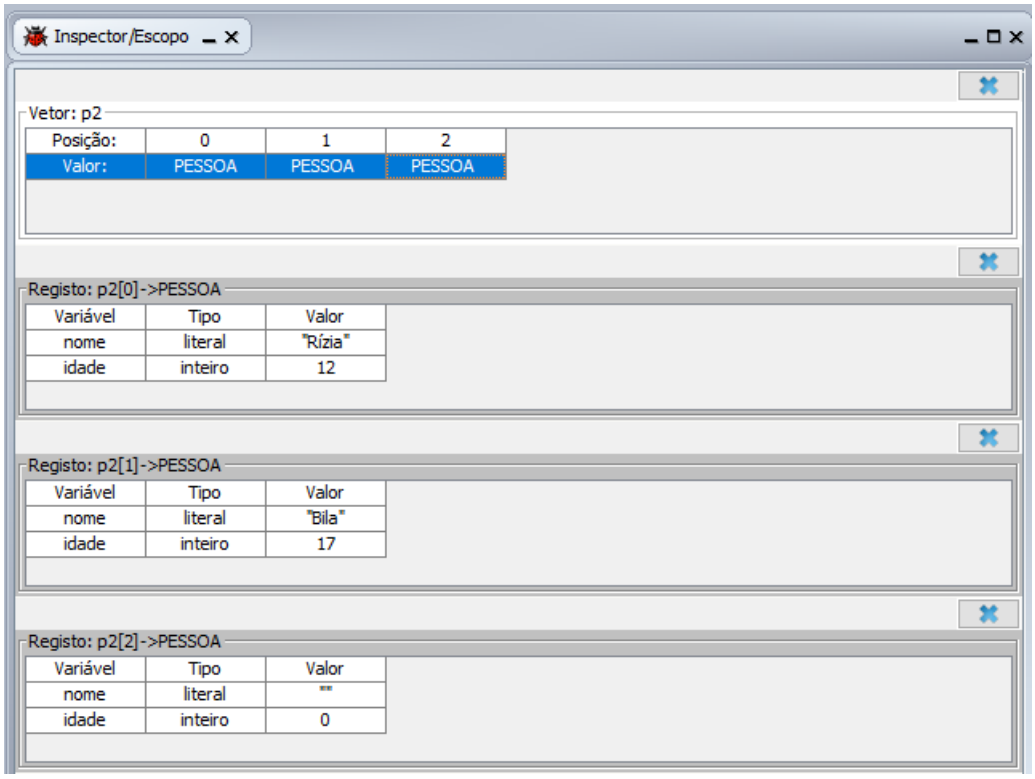


Figura 6.11: Inspetor de Estruturas de Dados Durante a Execução Passo a Passo

- **Inspetor Individual de Estruturas de Dados**

  Na Figura 6.11 apresentamos a inspeção individual de cada estrutura de dados do programa, nesse caso estamos considerando o mesmo programa fonte apresentado na Figura 6.10.

- **Visualizador de Condicionais**

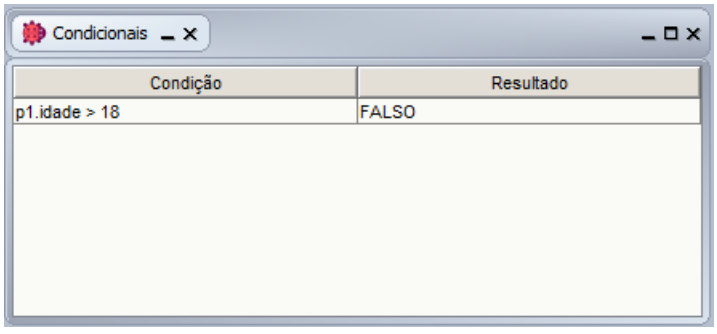


Figura 6.12: Visualização das Condicionais do Programa

Na Figura 6.12, apresentamos a funcionalidade do visualizador de condicionais do programa fonte. Durante a execução um condicional pode tomar estados

diferentes. No caso da Figura 6.12 o estado da condição avaliada é FALSA, tendo em consideração que estamos tratando do mesmo programa já apresentado na Figura 6.10.

6.4 *Profiling* de Métodos e Variáveis

Esta funcionalidade consiste em obter a quantidade de chamadas efetuadas a um determinado método e a quantidade de variáveis conforme o seu tipo de dados, a partir do código fonte. Considere a Figura 6.14 com o código fonte. Após a execução, a Figura 6.15 apresenta o resultado do *Profiling* de métodos e variáveis.

A figura 6.13, indica onde o **profiler** é ativado.

Figura 6.13: Ativação do *Profiling*

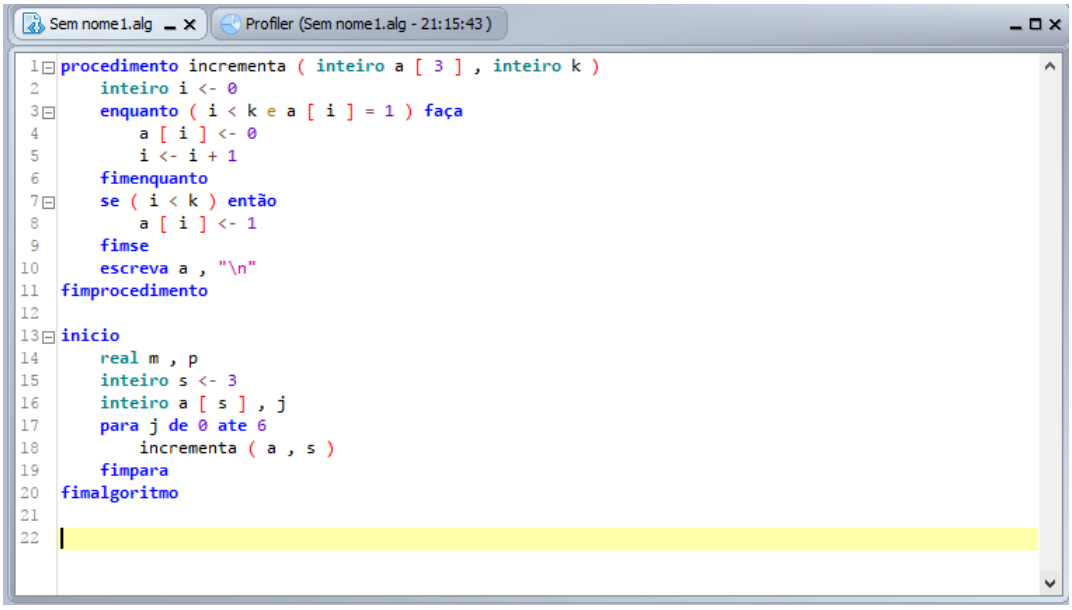


Figura 6.14: Um Pequeno Programa "contador"

6.5 Assinaturas de Métodos

É uma funcionalidade muito importante quando pretendemos utilizar funções e procedimentos. Esta funcionalidade permite que o programador consiga utilizar um método

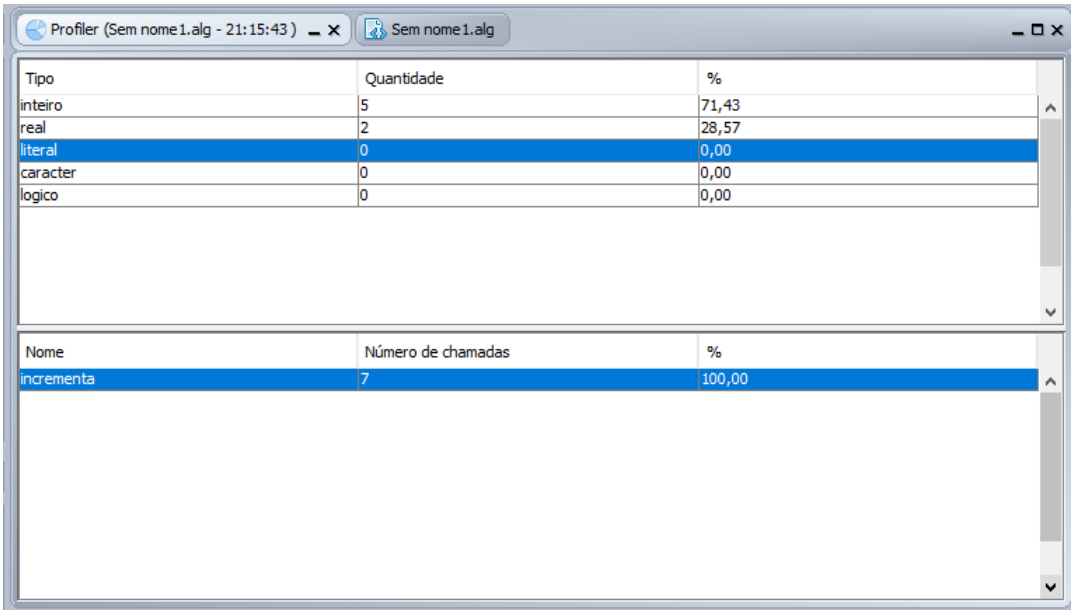


Figura 6.15: Funcionamento do *Profiling*

mesmo sem ter criado antes. Para que isso funcione é necessário que "assine antes". A Figura 6.16 apresenta a utilização desta funcionalidade proporcionada pela nossa linguagem.

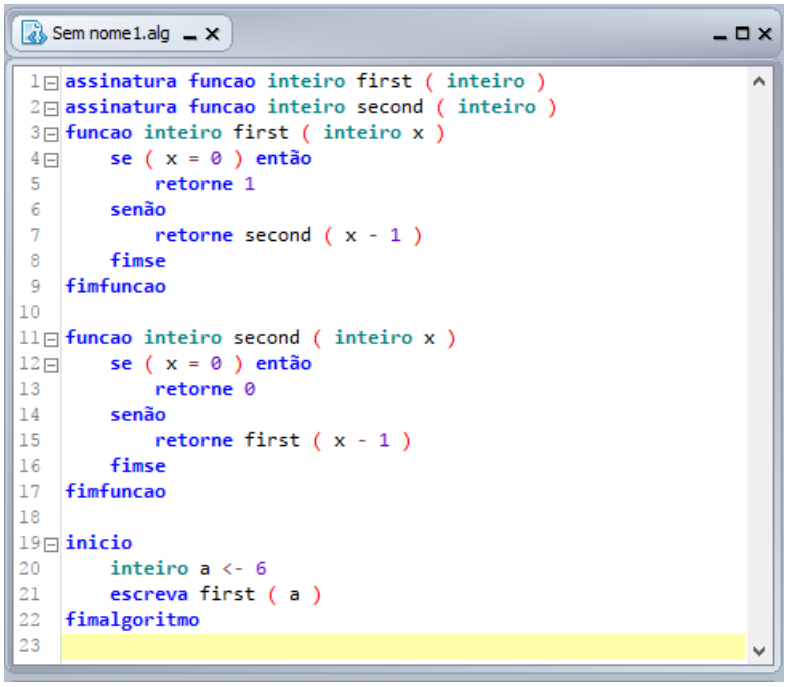


Figura 6.16: Utilização de Assinaturas

No topo da Figura 6.16, vemos duas declarações de assinatura (linhas 1 e 2 respetiva-

mente). Em seguida, na construção do método "first" o método "second" é chamado, porém na construção do método "second" o método "first" é chamado, ou seja existem chamadas de métodos que ainda não foram construídos, mas a nossa linguagem permite através do uso de assinaturas.

6.6 Síntese

Apresentamos as funcionalidades mais relevantes da nossa ferramenta. Começamos por mostrar o potencial do nosso editor de código e as funcionalidades da nossa consola (ver Secção 6.1). Posteriormente vimos que podemos imprimir na consola uma estrutura de dados complexo (ver Secção 6.2. Em seguida vimos a execução do nosso programa fonte passo a passo (ver Secção 6.3) com um exemplo claro. Finalizamos mostrando a funcionalidade *profiling* de métodos e variáveis, que gera uma estatística da quantidade de vezes que um método é chamado e a quantidade de variáveis foram utilizadas no programa fonte (ver Secção 6.4) e Assinaturas de métodos que consiste em chamar métodos não criados, desde que seja assinada no início do programa-fonte (ver Secção 6.5).

Conclusões e Perspetivas 7

Neste último capítulo dedicamos-nos a sumarizar todo trabalho que foi desenvolvido nesta tese. Inicialmente destacaremos os principais contributos deste árduo trabalho e posteriormente abordaremos sobre as possibilidades de trabalho futuro.

7.1 Principais Contribuições

O trabalho desenvolvido nesta tese focou-se essencialmente na construção de uma ferramenta com cariz de software educativo, que ajudasse estudantes a aprender e/ou ensinar a lógica de programação. Vimos ao longo de todo trabalho vários problemas inerentes ao ensino e aprendizagem da lógica de programação, problemas estes que tratamos de estudar e com ajuda da nossa ferramenta, melhorar o processo e até propor uma nova metodologia de ensino e aprendizagem.

As principais contribuições deste trabalho são:

- **A melhoria do entendimento da lógica dos programas**. Durante as nossas pesquisas descobrimos que os estudantes melhoram o entendimento da programação quando conseguem visualizar, não apenas o resultado final dos seus programas mas essencialmente o conteúdo das variáveis. Tal problema foi resolvido neste trabalho onde incluímos visualizadores de conteúdos de variáveis simples, vetores e também visualizadores de resultados das avaliações condicionais do programa. Foram implementados também, inspetores dedicados a visualizar de forma individual as estruturas de dados complexas (matrizes e registos).
- **Execução passo a passo e *Break Point***. A execução do programa e apresentação do resultado final por si só não são suficientes para uma didática eficaz, portanto incluímos na nossa ferramenta a possibilidade de executar o programa linha por linha, para que o estudante aprendiz possa perceber a sequência lógica do seu programa bem como o estado das variáveis após a execução de uma linha

de código. Para quem ensina, torna-se uma funcionalidade interessante para concretizar vários conceitos de programação. O *Break Point* foi projetado para permitir parar programas a meio do programa, possibilitando assim um estudo minucioso do código, podendo passar várias linhas até à linha desejada sem que percorra uma de cada vez.

- **Estatística de métodos e variáveis** (*profiling*). Uma contribuição importante no nosso trabalho, consiste em obter o número de vezes que os métodos foram chamados durante a execução do programa e a quantidade de variáveis que são utilizadas na memória, classificados pelo seu tipo de dados.

- **Uso de métodos com pré-assinaturas**. O uso de métodos foi melhorado com esta funcionalidade, que permite invocar métodos mesmo que não sejam criados. É importante para o aprendiz da lógica de programação ter desde logo essa experiência, porque linguagens de programação como C e derivados utilizam esse mecanismo.

7.2 Trabalho Futuro

Com essa tese foi notório observar os grandes desafios que essa área nos propõem, e alguns desses desafios levam muitos anos e um esforço redobrado, tanto mental como físico.

Como perspetiva achamos que, seria importante que em trabalhos futuros se possam implementar mais estruturas de dados para promover um estudo maior e mais completo dos estudantes aprendizes, bem como dar mais possibilidade aos ensinadores na utilização da ferramenta como um auxiliar completo e eficaz. A possibilidade de implementar estudos completos em forma de pequenos cursos em vários níveis e pedagogicamente aceites, para um estudo planeado por parte do estudante. Esperamos que nas próximas versões possa-se ter uma dedicação muito especial ao módulo de conversões da nossa pseudo-linguagem para as linguagens de auto-nível para que o aprendiz possa cada vez mais estar próximo das linguagens reais e de produtividade como o Java, Objetive C, CSharp, entre outras.

Em suma, espera-se que o nosso ambiente de desenvolvimento integrado para ensino e aprendizagem de programação não pare por aqui e que mais investigadores e desenvolvedores possam encarar esse desafio de forma séria para a melhoria do ensino em Angola e no resto dos PALOPs.

E por fim, a construção deste tipo de ferramenta não deveria ser encarada, ou associada ao desenvolvedor soldado medieval que luta com o grande dragão da complexidade. Pelo menos no meio académico essa atividade devia ser tão inofensiva e tão agradável quanto alimentar centenas de pombos numa praça.

Referências

[And13] Eduardo Martins Andreo, Joao Ricardo & Morgado. Um estudo sobre a utilização do software scratch como ferramenta de apoio ao ensino da disciplina de lógica de programação. *Mímesis*, pages 45–62, 2013.

[Ant08] Professor Antônio. Visualg3.0 - o melhor editor e interpretador de algoritmos é do brasil, 2008. [Online; acessado em 20-Setembro-2017].

[Bar09] Liane & Bercht Magda Barcelos, Ricardo & Tarouco. O uso de mobile learning no ensino de algoritmos. *RENOTE*, 7(3):327–337, 2009.

[Bar14] D. I. S. & Costa E. B. Barbosa, A. A. & Ferreira. *Influência Da Linguagem No Ensino Introdutório De Programação*. II CBIE 2014, 2014.

[Cam09] RLBL Campos. Lógica de programação: Há como melhorar o aprendizado fugindo dos padrões estabelecidos nos livros didáticos e adotados pela maioria dos docentes? In *XVII Congresso Iberoamericano de Educación Superior em Computacion (CLEI-2009-CIESC)*, pages 22–25, 2009.

[Cam10] RLBL Campos. Como pode ser a aplicação da metodologia erm2c para a melhoria do processo ensino-aprendizado de lógica de programação? *Paraguai: XVII Congresso Iberoamericano de Educación Superior em Computacion (CLEI-2010-CIESC)*, 2010.

[COD01a] CODEWITZ. Codewitz / minerva - for better programming skills (109986-cp-1-2003-1-fi-minerva-mpp), 2001. [Online; acessado em 19-Setembro-2017].

[COD01b] CODEWITZ. Codewitz asia-link – cooperation for better programming skills (bd asia-link/10/095-229), 2001. [Online; acessado em 19-Setembro-2017].

[COD01c] CODEWITZ. Codewitz network web site, 2001. [Online; acessado em 19-Setembro-2017].

REFERÊNCIAS

[Cor02] Charles Eric & Rivest Ronald & Stein Clifford Cormen, Thomas H. & Leiserson. *Introduction to Algorithms - Second Edition*. The MIT Press, 2002.

[CRM15] Ruan Carvalho, Pedro Rosa, and José Geraldo Ribeiro & Costa Gabriella Machado, João Vítor & Júnior. Ferramenta para auxílio na aprendizagem de lógica de programação em sistemas informatizados. In *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, volume 4, page 1144, 2015.

[Dé98] Gerwin Klein & Steve Rowe & Régis Décamps. Jflex, 1998. [Online; acessado em 10-Outubro-2017].

[Del04] José Antonio Moreira & SOUZA Izabel F & Campos Marcio & Rapkiewicz Clevi Elena Delgado, Carla & Xexeo. Uma abordagem pedagógica para a iniciação ao estudo de algoritmos. In *XII Workshop de Educação em Computação*, 2004.

[Dev12] Muatsoft Developer. Rsyntaxtextarea - a syntax highlighting text component, 2012. [Online; acessado em 10-Outubro-2017].

[Dev14] Muatsoft Developer. Angocompiler software, 2014. [Online; acessado em 10-Outubro-2017].

[DJ09] Gláucia Silva De Jesus, Andreia & Brito. Concepção de ensino-aprendizagem de algoritmos e programação de computadores: a prática docente. *Varia Scientia*, 9(16):149–158, 2009.

[dR88] Heloísa Vieira da Rocha. O uso da linguagem logo em um curso de introdução à programação de computadores para alunos ingressantes no bacharelado de ciência da computação. *Memos do NIED*, 4(10), 1988.

[dS06] Heitor Augustus Xavier dos Santos, Rodrigo Pereira & Costa. Análise de metodologias e ambientes de ensino para algoritmos, estruturas de dados e programação aos iniciantes em computação e informática. *INFOCOMP*, 5(1):41–50, 2006.

[Dua10] Hugo & Mello Thiago Silva Duarte, Alexandre & Moreira. Competitividade como fator motivacional para o estudo de computaçao. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 1, 2010.

[Fer02] Jorge Henrique Cabral Fernandes. Ensino introdutório de computação e programação: Uma abordagem concreta, construtivista, linguística e histórica. In *X Workshop de Educação em Computação–WEI*, 2002.

[FER08] CRISTIAN FERRARI, FABRICIO & CECHINEL. Introdução a algoritmos e programação. *Bagé: Universidade Federal do Pampa*, 2008.

[For05] A. L. V. Forbellone. *Lógica de programação: a construção de algoritmos e estruturas de dados*. Pearson Prentice Hall, 2005.

[Gom00] A. J. Gomes. Ambiente de suporte à aprendizagem de conceitos básicos de programação. Master's thesis, 2000.

[Hol90] A. I. Holub. *Computer Design in C*. Prentice-Hall, Inc. New Jersey, 1990.

[Inf98a] InfoNode. Infonode - docking windows developer's guide, 1998. [Online; acessado em 10-Outubro-2017].

[Inf98b] InfoNode. Infornode - easy to use yet powerful java componentes, 1998. [Online; acessado em 10-Outubro-2017].

[Jún05] Clevi Elena & Delgado Carla & Xexeo José Antonio Moreira Júnior, JCRP & Rapkiewicz. Ensino de algoritmos e programação: uma experiência no nível médio. In *XIII Workshop de Educação em Computação (WEI'2005). São Leopoldo, RS, Brasil*, page 12, 2005.

[Jún15] Bruno Batista Júnior, Rogério Paulo Marcon & Boniati. Logicblocks: Uma ferramenta para o ensino de lógica de programação. *Anais do EATI-Encontro Anual de Tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação*, pages 63–70, 2015.

[MAC12] MACP. Macp - compilador portugol, 2012. [Online; acessado em 20-Setembro-2017].

[Mar03a] L. M. M Marczak, S. dos S. & Giraffa. Ambientes inteligentes para suporte ao ensino de programação. Master's thesis, 2003.

[Mar03b] LH de A Martins, Scheila Wesley & Correia. O logo como ferramenta auxiliar no desenvolvimento do raciocínio lógico–um estudo de caso. In *Internacional Conference on Engineering and Computer Education–ICECE*, 2003.

REFERÊNCIAS

[mig08] miguelcz. Portugol viana project, 2008. [Online; acessado em 20-Setembro-2017].

[Mot08] Lis W Kanashiro & Favero Eloi Luiz Mota, Marcelle Pereira & Pereira. Javatool: Uma ferramenta para o ensino de programação. In *Congresso da Sociedade Brasileira de Computação. Belém. XXVIII Congresso da Sociedade Brasileira de Computação*, pages 127–136. SBC, 2008, 2008.

[Nos14] Fillipi & Raabe A & others Noschang, Luis Fernando & Pelz. Portugol studio: Uma ide para iniciantes em programaçao. pages 535–545, 2014.

[Ren09] S & Janakiram D Renumol, VG & Jayaprakash. Classification of cognitive difficulties of students to learn computer programming. *Indian Institute of Technology, India*, 2009.

[Sal10] Vanessa Farias Sales, Chrystian Gesteira & Dantas. Progame: um jogo para o ensino de algoritmos e programação. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 1, 2010.

[San03] R. Santos. *Introdução à programação orientada a objetos usando Java*. Campus, 2003.

[VIA08] PORTUGOL VIANA. Portugolviana, 2008. [Online; acessado em 20-Setembro-2017].