Universidade Federal do Rio Grande do Norte

Centro de Tecnologia

Departamento de Computação e Automação

Algoritmo e Lógica de Programação

Conceitos de Linguagens de Programação

DCA 800 – Eng. Química

Abril / 2004

SUMÁRIO

1. LINGUAGENS DE PROGRAMAÇÃO

Linguagens de programação são usadas para descrever algoritmos; isto é, seqüências de passos que levam à solução de um problema. Uma linguagem de programação precisa suportar a definição de ações e prover meios para especificar operações básicas de computação, além de permitir que os usuários especifiquem como estes passos devem ser seqüenciados para resolver um problema. Uma linguagem de programação pode ser considerada como sendo uma notação que pode ser usada para especificar algoritmos com precisão.

1.1 Classificação das Linguagens de Programação

As linguagens de programação podem ser agrupadas em dois grandes grupos: linguagens de baixo nível e linguagens de alto nível.

As *linguagens de baixo nível* são restritas a linguagem de máquina e tem uma forte relação entre as operações implementadas pela linguagem e as operações implementadas pelo hardware.

As *linguagens de alto nível* por outro lado, aproximam-se das linguagens utilizadas por humanos para expressar problemas e algoritmos. Cada declaração numa linguagem de alto nível equivale a várias declarações numa linguagem de baixo nível.

A vantagem principal das linguagens de alto nível é a *abstração*. Isto é o processo em que as propriedades essenciais requeridas para a solução do problema são extraídas enquanto esconde os detalhes da implementação da solução adotada pelo programador. Com o nível de abstração aumentado, o programador pode concentrar-se mais na solução do problema ao invés de preocupar-se como o hardware vai tratar do problema.

No início da computação os programadores eram obrigados a programar usando linguagem de máquina, que nada mais é do que seqüências de dígitos binários (0s e 1s). Por exemplo, a instrução "incremente o valor no acumulador" deveria ser representada como:

```
10100100
```

Ou na melhor hipótese, escrito sob a forma de um número hexadecimal

```
A4
```

É claro que esta representação tinha muitas desvantagens:

- Há uma grande probabilidade de erro em todos os estágios do processo de programação.
- A programação mesmo sendo com algoritmos simples resulta em longos programas, o que dificulta o processo de validação e detecção de erros.
- O cálculo de endereços de memória devem ser feitos manualmente, com um árduo trabalho e uma grande probabilidade de erros.

Algumas das desvantagens podem ser superadas fazendo com que o computador seja o responsável pelo estágio de tradução. O programa ainda é escrito em termos de operações básicas de máquina, mas a tradução em código binário é feita pelo computador. O programa que faz essa tradução é chamado de *assembler*. Até mesmo o mais simples dos *assemblers* modernos podem

reconhecer endereçamentos simbólicos e *mnemônicos* representado operações de máquina. Assim, por exemplo, o programador precisa somente escrever:

```
ADD ival
```

Para especificar uma instrução para adicionar o conteúdo de localização `ival` para o acumulador. O *assembler* então faz a tradução para a string equivalente de 0s e 1s. O *assembler* também trata do problema de cálculo de endereço, usando nomes em formato de texto para endereçar os dados. A conseqüência desta automação de tradução é que os programas em linguagem Assembly são muito mais fáceis de escrever e depurar que programas em linguagem de máquina.

1.1.1 Linguagens de Alto Nível Comparadas com Linguagens de Baixo Nível

As linguagens de alto nível podem oferecer muito mais vantagens que as linguagens de baixo nível. A principal motivação para o uso de linguagens de alto nível é que os problemas podem ser solucionados muito mais rapidamente e com muito mais facilidade, pois apresenta um considerável numero de tipos de dados definidos, além das facilidades da programação estruturada.

Os programas em linguagem de alto nível são muito mais fáceis de serem desenvolvidos, entendidos e depurados por diversas razões:

- Elas são mais prováveis de serem auto-documentadas.
- A estrutura do programa pode ser desenvolvida para refletir a estrutura do problema original.
- Nomes significativos podem ser escolhidos para variáveis e subprogramas.
- A solução do problema não necessita ser obscurecida pelo nível de detalhes necessários em um programa em linguagem de baixo nível.
- O programa em linguagem de alto nível é normalmente fácil de seguir e entender cada passo da execução.
- O compilador, nas linguagens de alto nível, normalmente provê facilidades para a depuração, como visualização dos valores das variáveis, dos registradores e da pilha. Além disso o compilador pode incluir instruções na geração de código para detectar erros em tempo de execução, como overflow numéricos e violação de limites de vetores e matrizes.
- A utilização de linguagens de baixo nível é indicada para funções que precisam implementar instruções de máquina específicas que não são suportadas por linguagens de alto nível, embora a grande maioria das linguagens de alto nível apresentam uma biblioteca que permite implementar instruções de baixo nível diretamente em seus programas.
- A grande eficiência e o reduzido tamanho dos programas desenvolvidos em linguagens de baixo nível são as principais vantagens dessas linguagens.

1.2 Histórico das Linguagens de Programação

Existem centenas de linguagens de programação, desenvolvidas desde o início da computação. Essas linguagens foram agrupadas de acordo com suas e características e época em que foram desenvolvidas em 4 gerações:

1.2.1 Linguagens de primeira geração

A primeira geração de linguagens remonta aos dias da codificação em linguagem de máquina, surgidas com o início da computação na década de 50, especificamente de 1950 a 1962. A Linguagem de máquina e Assembly representam esta primeira geração das linguagens de programação.

Essas linguagens totalmente dependentes da máquina, exibem o mais baixo nível de abstração que uma linguagem pode ser representada.

Essas linguagens somente devem ser usadas quando as linguagens de mais alto nível não satisfizerem as necessidades ou não forem suportadas.

1.2.2 Linguagens de Segunda Geração

A segunda geração de linguagens de programação foi desenvolvida de 1962 a 1974 e serviu de base para o desenvolvimento das modernas linguagens de programação.

As caraterísticas marcantes das linguagens de segunda geração foram o amplo uso com grande familiaridade e aceitação no mercado e a grande quantidade de bibliotecas de software, permitiram a programação multi-usuário, sistemas de execução em tempo real e desenvolvimento de gerenciadores de base de dados.

As linguagens Fortran, Cobol, Algol e algumas extensões como Basic, foram os representantes dessa segunda geração.

Fortran é uma linguagem ainda muito utilizada na área de engenharia e pela comunidade científica. Cobol é uma linguagem que foi aceita e ainda continua em uso para aplicações comerciais. Algol foi o precursor de muitas linguagens de terceira geração, por oferecer ricamente estruturas de controle e tipos de dados. Basic foi uma linguagem originalmente criada para o aprendizado e teve seu uso bastante reduzido já na década de 70.

1.2.3 Linguagens de Terceira Geração

As linguagens de terceira geração também chamadas de linguagens de programação modernas ou estruturadas, são caracterizadas pela grande capacidade procedural e estrutural de seus dados foram desenvolvidas de 1974 a 1986.

As linguagens de terceira geração tiveram como principais características a possibilidade de criar sistemas distribuídos, incorporar recursos mais inteligentes, e exigir um hardware menos robusto. Podem ser divididas em duas grandes categorias: linguagens de propósito geral e linguagens especializadas.

As linguagens de propósito gerais foram desenvolvidas baseadas principalmente na linguagem Algol e servem para uma infinidade de aplicações envolvendo desde a área científica, até

a área comercial. As linguagens C, Pascal, PL/1 e Modula-2 são as principais linguagens desta categoria, sendo que as duas primeiras continuam bastante usadas atualmente.

As linguagens especializadas são caracterizadas pela forma sintática não usual com que foram desenvolvidas para uma aplicação distinta. Centenas de linguagens especializadas estão em uso atualmente. Dentre as linguagens que encontram aplicações na área de engenharia de software podemos destacar a linguagem Lisp desenvolvida especialmente para manipular símbolos e listas, Prolog desenvolvida para tratar e representar conhecimentos,. Smalltalk criada para representar os dados em forma de objetos, sendo a primeira a ser puramente orientada a objetos, APL desenvolvida para manipular vetores, e a linguagem Forth desenvolvida para desenvolver softwares para microprocessadores.

1.2.4 Linguagens de quarta geração

A quarta geração das linguagens de programação foram desenvolvidas a partir de 1986 e teve como características principais a geração de sistemas especialistas, o desenvolvimento de inteligência artificial e a possibilidade de execução dos programas em paralelo.

No decorrer da história temos percebido uma evolução para uma abstração maior na geração de programas, usando linguagens de mais alto nível.

A primeira geração de linguagens de programação trabalhavam com um reduzido conjunto de instruções a nível de máquina. A segunda e terceira geração de linguagens de programação foram desenvolvidas num nível que representam os programas computacionais, distinta e independentemente da arquitetura do processador, mas com completa descrição detalhada dos procedimentos algorítmicos do programa. Com o passar do tempo, as linguagens de quarta geração foram desenvolvidas com um nível de abstração ainda mais alto.

As linguagens de quarta geração, conhecidas também como linguagens artificiais contém uma sintaxe distinta para representação de estruturas de controle e dos dados. Essas linguagens por combinarem características procedurais e não procedurais, representam estas estruturas com um alto nível de abstração, eliminando a necessidade de especificar algoritmicamente esses detalhes.

As linguagens de quarta geração podem ser classificadas em três categorias: linguagens de consulta, geradoras de programas e outras linguagens (4GL).

As linguagens de consulta foram desenvolvidas para manipular bases de dados, permitindo o gerenciamento de um grande número de informações armazenados em arquivos.

As linguagens geradoras de programas representam uma sofisticada classe das linguagens 4GL. Permitem ao usuário ou programador criar facilmente programas complexos em linguagens de terceira geração, utilizando bem menos declarações e comandos. Estas linguagens possuem um nível bem mais alto que as de terceira geração.

Enquadradas como outras linguagens de quarta geração temos as linguagens usadas em sistemas de apoio à decisão, linguagens utilizadas para modelagem de sistemas, linguagens de prototipação, e linguagens de especificação formal que produzem código de máquina.

1.2.5 Linguagens desenvolvidas

No decorrer da história da computação centenas de linguagens foram desenvolvidas. Já em 1972 haviam mais de 200 linguagens desenvolvidas, sendo que a maioria era para objetivos

específicos ou acadêmicos, sendo que dessas apenas 12 podem ser consideradas como importantes e significativas.

Relação das linguagens de programação com o ano em que foram desenvolvidas:

1957	FORTRAN	
1958	ALGOL	
1960	LISP	
1960	COBOL	
1962	APL	
1962	SIMULA	
1964	BASIC	
1964	PL/1	
1966	ISWIM	
1970	Prolog	
1972	C	

1975	Pascal	
1975	Scheme	
1977	OPS5	
1978	CSP	
1978	FP	
1980	dBase II	
1983	Smalltalk	80
1983	Ada	
1983	Parlog	
1984	Standard ML	
1986	C++	

1986	CLP(R)	
1986	Eiffel	
1988	CLOS	
1988	Mathematica	
1988	Oberon	
1990	Haskell	
1995	Delphi	
1995	Java	

1.2.6 Novas direções das linguagens de programação

As novas direções das linguagens de programação, especialmente das linguagens de quarta geração é a sua aplicação com metodologias orientadas a objetos. Essas linguagens são baseadas nos conceitos de objetos, que agrupam comandos de programação com dados em objetos que podem ser usados o tempo todo durante a execução do programa, o que é muito útil em ambientes de execução paralela.

Outra tendência dos ambientes de desenvolvimento de programas é fornecer além das linguagens de programação, um ambiente de geração automática de código, onde o programador especifica através de ferramentas visuais as características do programa e a ferramenta se encarrega de gerar a codificação na linguagem específica. Estas ferramentas são muito difundidas na programação para Windows, e são também chamados de RAD (Desenvolvimento Rápido de Aplicativos).

A nova geração das linguagens de programação, que já é chamada por muitas pessoas de quinta geração, é baseada em métodos de consulta e utilizam comandos escritos em linguagens naturais, permitindo uma fácil comunicação com o computador.

1.2.7 Processadores de Linguagens

As linguagens de alto nível são as linguagens que possuem uma certa independência da máquina, pois não são desenvolvidas utilizando instruções especificas do processador (linguagem de máquina), mas um conjunto de comandos que são transformados em linguagens de máquina.

As linguagens de programação são implementadas por *compilação* de programas em linguagem de máquina, por *interpretação* das mesmas, ou por alguma combinação de *compilação* e *interpretação*.

Qualquer sistema para processamento de programas – executando-os ou preparando-os para a execução – é chamado *processador de linguagem*. Processadores de linguagem incluem compiladores, interpretadores, e ferramentas auxiliares como editores dirigidos à sintaxe.

1.3 Processamento de Linguagens

Embora seja teoricamente possível a construção de computadores especiais, capazes de executar programas escritos em uma linguagem de programação qualquer, os computadores existentes hoje em dia são capazes de executar somente programas em uma linguagem de nível baixo, a *linguagem de máquina.* Linguagens de máquina são projetadas em função da rapidez de execução de programas, do custo de sua implementação e da flexibilidade com que permitem a construção de programas de nível mais alto. Por outro lado, linguagens de programação são freqüentemente projetadas em função da facilidade na construção e da confiabilidade de programas. Um problema básico, então, é como uma linguagem de nível mais alto pode ser implementada em um computador cuja linguagem de máquina é bastante diferente, e de nível bem mais baixo.

Existem basicamente duas alternativas para esta implementação: interpretação e tradução.

1.3.1 Interpretação

Nesta solução, as ações indicadas pelos comandos da linguagem são diretamente executadas. Em geral existe para executar cada ação possível um subprograma (escrito na linguagem de máquina do computador hospedeiro). Assim, a interpretação de um programa é feita pela chamada daqueles subprogramas, em uma seqüência apropriada.

Mais precisamente, um interpretador é um programa que executa repetidamente a seguinte seqüência:

1. Obter o próximo comando do programa.
2. Determinar que ações devem ser executadas.
3. Executar estas ações.

Esta seqüência é bastante semelhante àquela executada por computadores tradicionais, a saber:

1. Obter a próxima instrução (aquela cujo endereço é especificado no indicador de instruções da máquina).
2. Deslocar o indicador de instruções (obtendo o endereço da próxima instrução a ser executada).
3. Decodificar a instrução.
4. Executar a instrução.

Esta semelhança mostra que a interpretação pode ser encarada como a simulação, em um computador hospedeiro, de uma máquina especial cuja linguagem de máquina é a linguagem de nível mais alto.

1.3.2 Tradução

Nesta solução, programas escritos em linguagem de alto nível são traduzidos para versões equivalentes em linguagem de máquina, antes de serem executados. Esta tradução é feita em vários passos. Por exemplo, subprogramas podem ser inicialmente traduzidos para código Assembly, este pode depois ser traduzido para código relocável (objeto), em linguagem de máquina; em seguida, unidades em código relocável (objeto) podem ser ligadas em uma única unidade relocável (um único código objeto); e, finalmente, o programa inteiro é carregado na memória principal, como código executável de máquina. Os tradutores usados em cada um destes passos tem nomes especiais: compilador, montador, ligador (*linker*) e carregador, respectivamente.

Em alguns casos, a máquina onde a tradução é feita (a *máquina hospedeira)* é diferente daquela onde o código gerado é executado (a *máquina objetivo)*. Neste caso o processo é chamado *tradução cruzada.* Tradutores cruzados são a única opção de tradução quando a máquina objetivo é muito pequena para conter o tradutor.

A interpretação pura e a tradução pura são dois extremos. Na prática, muitas linguagens são implementadas por uma combinação destas técnicas. Um programa pode ser traduzido para um código intermediário, que é então interpretado. Este código intermediário pode ser simplesmente uma representação formatada do programa-fonte, de onde foi removida informação irrelevante (como comentários e espaços) e onde os componentes de cada comando estão armazenados em formato fixo, de maneira a simplificar a decodificação de instruções que se segue. Neste caso, a solução é basicamente interpretativa. Alternativamente, o código intermediário poderia ser o código de máquina (de baixo nível) de uma máquina virtual que seria depois interpretada por programas. Esta solução, que depende mais fortemente de tradução, pode ser adotada na geração de código portátil, isto é, código mais facilmente transferível para outras máquinas do que código em linguagem de máquina.

Em uma solução puramente interpretativa, a execução de um comando pode requerer um processo de decodificação bastante complicado, para determinar as operações a serem executadas e seus operandos. Na maioria dos casos, a mesma decodificação é executada cada vez que o comando é encontrado. Conseqüentemente, se o comando aparece em um trecho de programa executado com freqüência (por exemplo, em uma repetição interna), a decodificação idêntica afeta sensivelmente a rapidez de execução do programa.

Por outro lado, na tradução pura, o código de máquina é gerado para cada comando de alto nível. Neste caso o tradutor decodifica cada comando somente uma vez. Os componentes usados com freqüência são então decodificados, na sua representação em linguagem de máquina, várias vezes; como isto é feito eficientemente por circuitos internos, a tradução pura pode economizar tempo de execução em comparação à interpretação pura.

Por outro lado, é possível que a interpretação pura economize memória. Na tradução pura, cada comando de alto nível pode ser traduzido para dezenas ou centenas de instruções de máquina. Em uma solução puramente interpretativa, os comandos de alto nível são mantidos em sua forma original, e as instruções necessárias à sua execução são guardadas em um subprograma do interpretador. A economia de memória é evidente se o programa é grande e usa a maioria dos comandos da linguagem. Por outro lado, se todos os subprogramas do interpretador são mantidos na memória principal durante a execução, o interpretador pode desperdiçar memória na execução de programas pequenos, que usam somente alguns comandos da linguagem.