Linguagens de Programação - Parte VI
Especificação de Linguagens de Programação (processamento, interpretação e tradução - ambientes de programação)

Especificação de Linguagens de Programação

PROCESSAMENTO (modelo de execução de linguagem)

Uma linguagem de alto nível é definida na forma de um conjunto de instruções, em princípio passíveis de execução por uma máquina.

Pode-se dizer que uma linguagem de programação define formalmente uma máquina de computação.

No entanto, como vimos, um computador real só pode executar programas expressos em uma linguagem definida pelo hardware, a linguagem de máquina.

UNIVASF

Especificação de Linguagens de Programação

PROCESSAMENTO (modelo de execução de linguagem)

Nenhum hardware (pelo menos no âmbito comercial) incorpora instruções com grau de complexidade de uma linguagem de alto nível.

A máquina capaz de executar programas de alto nível é então virtual, só existe formalmente, e a execução de um programa de alto nível sempre é um processo de simulação dessa maquina virtual.

UNIVASF

Especificação de Linguagens de Programação

PROCESSAMENTO (modelo de execução de linguagem)

O processador de uma linguagem é um sistema de computação cuja tarefa é receber um programa escrito nessa linguagem e criar condições para que as atividades prescritas por este programa sejam realizadas por um computador que o receber.

Dito de outra forma: um processador de linguagem resolve o problema de implementação da máquina virtual definida pela linguagem.

UNIVASF

Especificação de Linguagens de Programação

PROCESSAMENTO (modelo de execução de linguagem)

As soluções dadas a este problema podem ser consideradas do ponto de vista do usuário da linguagem (programador) ou do implementador do processador.

Para o escopo da disciplina, que norteia nosso estudo, é pertinente apenas o primeiro caso.

Quanto ao segundo, tem desenvolvimento maior em disciplinas que objetivam a implementação de linguagens.

UNIVASF

PROCESSAMENTO (modelo de execução de linguagem)

Modos de processamento

Há duas abordagens fundamentais quanto à implementação de linguagens: interpretação e tradução.

Visando distingui-las, vale uma metáfora adaptada da obra Dershem, Hebert L.; Jipping, Michael J. Programming languages: structures and models. Belmont: Wadsworth Publishing Co., 1990.

UNIVASF

PROCESSAMENTO (modelo de execução de linguagem)

Modos de processamento

Imagine um funcionário que fala inglês e seu chefe, que fala português.

Todos os dias o chefe prepara uma lista, em português, de tarefas a serem executas pelo funcionário.

Este usará um dicionário português-inglês para entender a lista e executar as tarefas.

O funcionário poderá adotar dois procedimentos.

UNIVASF

PROCESSAMENTO (modelo de execução de linguagem)

Modos de processamento

O primeiro é o que chamaremos abordagem *interpretativa*.

Na qual ele traduz a primeira instrução e executa a tarefa correspondente. Traduz a segunda instrução e executa a tarefa correspondente; E assim por diante, até completar o serviço.

UNIVASF

PROCESSAMENTO (modelo de execução de linguagem)

Modos de processamento

Na segunda abordagem — a tradução — o funcionário primeiramente traduz todas as instruções da lista de português para inglês, registrando as instruções traduzidas. Na sequência, toma a lista de instruções traduzidas e executa as tarefas prescritas.

UNIVASF

INTERPRETAÇÃO

Um interpretador traduz um comando (instrução) de um programa por vez e já invoca uma rotina que complete sua execução.

Interpretadores têm a vantagem de não traduzir instruções que nunca serão usadas.

Também consegue sempre relacionar ao longo do programa seus efeitos com as instruções correspondentes (isto é bom para observar um programa em funcionamento e detectar erros).

UNIVASF

Especificação de Linguagens de Programação

Normalmente, para uma mesma linguagem, tende a ser um programa menor que um compilador.

Tem como desvantagem principal o fato de a execução de programas ser mais lenta, devido à necessidade de retraduzir as instruções cada vez que seja necessário executar um programa. Ou mesmo durante uma única execução haverá retradução, sempre que for necessário repetir um trecho, seja numa repetição propriamente ou na invocação de um procedimento, por exemplo.

UNIVASF

TRADUÇÃO

Um tradutor produz, a partir do programa que lhe é fornecido (programa fonte), um programa equivalente, mas em uma linguagem executável (programa objeto).

O programa objeto pode ser diretamente executável (linguagem de máquina) ou estar em uma linguagem para a qual exista um processador, ao qual ainda será submetido o programa objeto.

UNIVASF

TRADUÇÃO

Conforme seja essa relação, entre linguagem fonte e linguagem objeto, os tradutores podem ser:

- montadores: traduzem de linguagem de baixo nível (assembly) para linguagem de máquina;
- compiladores: traduzem de linguagem de alto nível para linguagem de baixo nível (por exemplo, para linguagem de máquina);
- pré-processadores: traduzem de uma linguagem de alto nível para outra linguagem de alto nível.

UNIVASF

TRADUÇÃO

A grande vantagem dos tradutores é dar velocidade à execução dos programas, pois, o programa objeto resultante pode ser armazenado para posterior execução, sem necessidade de retradução.

Além disso, o tradutor pode, às custas de uma implementação mais complexa que a de um interpretador, fazer diagnósticos mais completos sobre as condições dos programas fontes, porque repassa todo o programa durante a tradução, antes de qualquer execução.

Isto é de particular valia para o programador no processo de depuração de seu programas.

SISTEMAS DE IMPLEMENTAÇÃO HÍBRIDOS

Alguns sistemas de implementação de linguagens são um meio termo entre os compiladores e interpretadores puros; eles traduzem programas em linguagem de alto nível para uma linguagem intermediária projetada para permitir fácil interpretação.

Este método é mais rápido do que a interpretação pura e são chamados de sistemas de implementação híbridos.

Java possui implementações híbridas e esta característica permite sua adaptação às mais variadas máquinas.

Linguagens de Programação

ESTRUTURA BÁSICA DE UM PROCESSADOR DE LINGUAGENS

Como os compiladores são os processadores de linguagens mais comuns, é a sua estrutura que será abordada aqui.

A estrutura de outros processadores de linguagens são similares, diferindo apenas em detalhes particulares.

De um modo geral, um processador tem seu trabalho subdividido em três fases:

- Análise léxica;
- Análise sintática;
- Geração de código.

UNIVASF

Linguagens de Programação

ESTRUTURA BÁSICA DE UM PROCESSADOR DE LINGUAGENS

- Análise léxica - decomposição da cadeia de caracteres que entra representando uma sentença, com o objetivo de separá-la adequadamente em seus constituintes importantes.

Tais elementos são exemplificados por identificadores, palavras-chaves, constantes, operadores, delimitadores, etc. Cada símbolo desses, isolado pelo analisador léxico, é conhecido como *token*.

UNIVASF

Linguagens de Programação

ESTRUTURA BÁSICA DE UM PROCESSADOR DE LINGUAGENS

- Análise sintática - verificação da entrada recebida para ver se é válida, i.e., se é uma sentença da linguagem.

O analisador sintático recebe do analisador léxico uma sequência de símbolos primitivos (*tokens*) e determina se estão dispostos conforme especifica a gramática da linguagem.

Linguagens de Programação

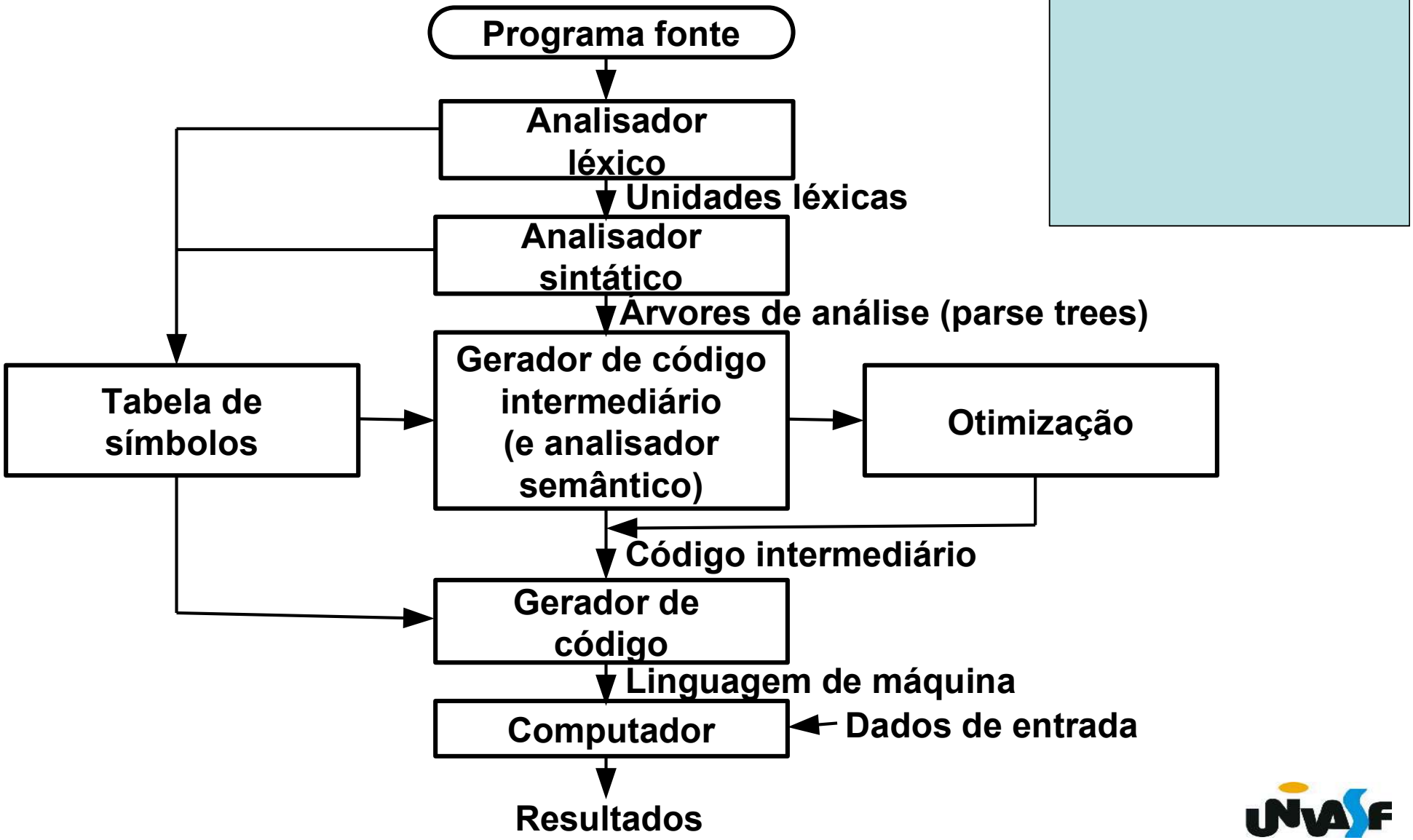
ESTRUTURA BÁSICA DE UM PROCESSADOR DE LINGUAGENS

- Geração de código - trata-se da produção de uma sequência de caracteres correspondente à tradução do programa-fonte, passível de execução posterior.

Uma forma de organizar estas fases é esquematizada na figura que veremos a seguir.

Cada fase produz uma versão transformada do programa em processamento, desde alguma forma de "código intermediário" até o código objeto final.

UNIVASF


Programa fonte
Analisador léxico
Unidades léxicas
Analisador sintático
Árvores de análise (parse trees)
Tabela de símbolos
Gerador de código intermediário (e analisador semântico)
Otimização
Código intermediário
Gerador de código
Linguagem de máquina
Computador
Dados de entrada
Resultados

AMBIENTES DE PROGRAMAÇÃO

Um ambiente de programação é o conjunto de ferramentas usadas no desenvolvimento de software.

Este conjunto pode se consistir em somente um sistema de arquivos, um editor de textos, um compilador e um *linkeditor*. Ou pode incluir uma grande coleção de ferramentas integradas, cada uma das quais acessada por meio de uma interface uniforme, a esta modalidade também se dá o nome de Ambiente Integrado de Desenvolvimento (IDE).

UNIVASF

AMBIENTES DE PROGRAMAÇÃO

Um IDE pode fornecer ainda programas auxiliares para o desenvolvimento de software, como, por exemplo, um gerador de código.

O gerador de código é uma ferramenta que possui a capacidade de gerar código a partir de um determinado modelo de software. Que pode, por exemplo, estar contido em um diagrama de blocos.

Linguagem de Programação C

Breve histórico de “C”

- Criada por Dennis Ritchie;
- Em 1972;
- Centro de Pesquisas da Bell Laboratories;
- Para utilização no S.O. UNIX.

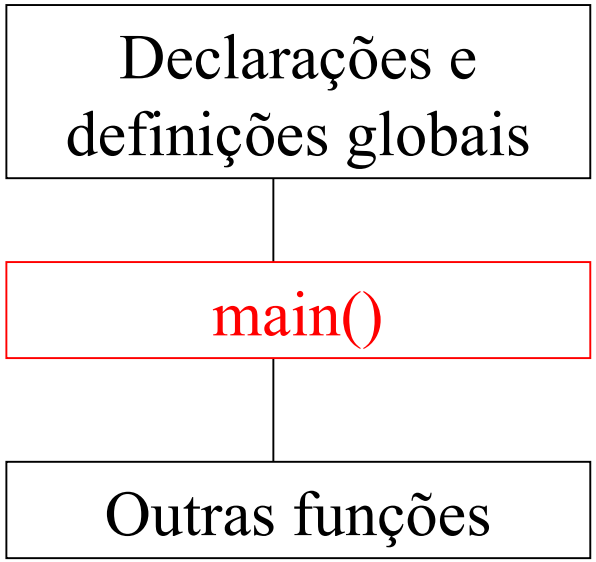
Características Básicas da Linguagem

- O C é uma linguagem de propósito geral;
- Case sensitive (sensível ao caso);
- Tipos de dados primitivos: caractere, inteiro e real;
- Possui estruturas de controle de fluxo para viabilizar a programação estruturada;
- Operadores aritméticos, lógicos, relacionais, condicionais e bit a bit;
- Funções de entrada e saída formatadas;
- Todo programa tem uma função chamada *main*();
- Praticamente, todo linha do programa termina com ";".

ANSI

Em 1983, o **Instituto Norte-Americano de Padrões** (**ANSI**) formou um comitê, X3j11, para estabelecer uma especificação do padrão da linguagem C. O padrão foi completo em 1989 e ratificado como ANSI X3.159-1989 “Programming Language C” (**C ANSI**).

Estrutura de um programa em C



Conceitos Básicos – Linguagem C

Constantes

Exemplos:

- Decimal (10, -23768)
- Hexadecimal (0x12,0x1fea28)
- Octal (0123)
- Real (2.34, 2.34E+5, 2.14E-9)
- Caractere (‘a’, ‘%’)

Identificadores

Identificadores são os nomes utilizados para referenciar variáveis, funções ou vários outros objetos definidos pelo programador. Sendo constituídos por:

- letras, dígitos e sublinhado(_);
- não podem começar com dígito;
- não podem ser iguais a uma palavra reservada e nem iguais a um nome de uma função disponibilizada pelo método utilizado para construção do programa.

UNIVASF

Palavras Reservadas e Comentários

- Palavras Reservadas

asm	const	else	for	sizeof	union
auto	continue	enum	goto	static	unsigned
break	default	extern	if	struct	void
case	do	far	int	switch	volatile
char	double	float	long	typedef	while

- Comentários

  /* este trecho não sera considerado pelo processador da linguagem */

UNIVASF

Tipos Primitivos

- **Caractere**
  - Definido por char;
  - Ocupa 8 bits (1 byte)
  - Faixa de valores: -128 à 127

- **Declaração de variáveis**
  - Exemplo de declaração de uma variável do tipo primitivo caractere:

```
char letra;
letra = 'A';
```

Operador de atribuição

UNIVASF

Tipos Primitivos

- **Inteiro**
  - Definido por int;
  - Ocupa 16 bits (2 bytes)
  - Faixa de valores: -32768 à 32767
  - Exemplo:

```
int num = 13;
num = num - 73;
```

UNIVASF

Tipos Primitivos

- **Ponto flutuante e ponto flutuante de precisão dupla**
  - float → 4 bytes
    - Seis dígitos de precisão
  - double → 8 bytes
    - Dez dígitos de precisão
  - faixa mínima de um valor em ponto flutuante
    - 1E-37 a 1E+37
  - Exemplo: float a,b,c=2.34;
    double x=2.38,y=3.1415,z;

UNIVASF

Operadores

- Operadores Aritméticos
  - Unários: +, -, ++, --

Exemplos: +1

-5

a=-b;

a++; ⇔ a=a+1;

a--; ⇔ a=a-1;

b=a++; ⇔ b=a;
a=a+1;

b=++a; ⇔ a=a+1;
b=a;

Obs.: a e b são variáveis numéricas.

UNIVASF

Operadores

- Operadores Aritméticos
  - Binários: +, -, *, /, %

10/3=? 3
10/3.0=? 3.3333
10.0/3=? 3.3333
10.0/3.0=? 3.3333

Não existe o operador \ (trabalhado em Introdução a Algoritmos)

Precedência (Hierarquia nas operações)

Hierarquia	Operação
1	Parênteses
2	Função
3	++,--
4	- (menos unário)
5	*, /, %
6	+, -

Operadores

- Operadores de Atribuição

  =, +=, -=, *=, /=, %=

  Exemplos:

  a=5;

  a+=5; ⇔ a=a+5;

  a*=5-2; ⇔ a=a*(5-2);

Operadores

Operadores Lógicos

Operador	Ação
&&	e
||	ou
!	não

&&	V	F
V	V	F
F	F	F

||	V	F
V	V	V
F	V	F

!V == F

Operadores

Operadores Relacionais

Operador	Ação
>	maior que
>=	maior ou igual a
<	menor que
<=	menor ou igual
==	igual a
!=	diferente de

Operadores

Precedência (Hierarquia dos operadores relacionais e lógicos)

Hierarquia	Operação
1	!
2	>, >=, <, <=
3	==, !=
4	&&
5	||

Operadores Lógicos bit a bit

Obs.: Aplicados a char e int.

Operador	Ação
&	AND(e)
|	OR(ou)
^	XOR(ou exclusivo)
~	NOT(não)
>>	Desloca os bits à direita
<<	Desloca os bits à esquerda

Exemplos:

```
  11000001   193 em binário
  01111111   127 em binário
& ________   AND bit a bit
  01000001   65 em binário

  10000000   128 em binário
  00000011   3 em binário
| ________   OR bit a bit
  10000011   131 em binário

  01111110   126 em binário
  01111001   121 em binário
^ ________   XOR bit a bit
  00000111   7 em binário
```

$1\ 1\ 0\ 0\ 0\ 0\ 0\ 1 =$

$2^7*1+2^6*1+2^5*0+2^4*0+2^3*0+2^2*0+2^1*0+2^0*1 =$

$128 + 64 + 0 + 0 + 0 + 0 + 0 + 1 = 193$

5 && 2 == 5 & 2

?

~00101100 byte original (44 em binário)
~11010011 após o 1º complemento
 00101100 após o 2º complemento

UNIVASF