Compiladores na visão do usuário

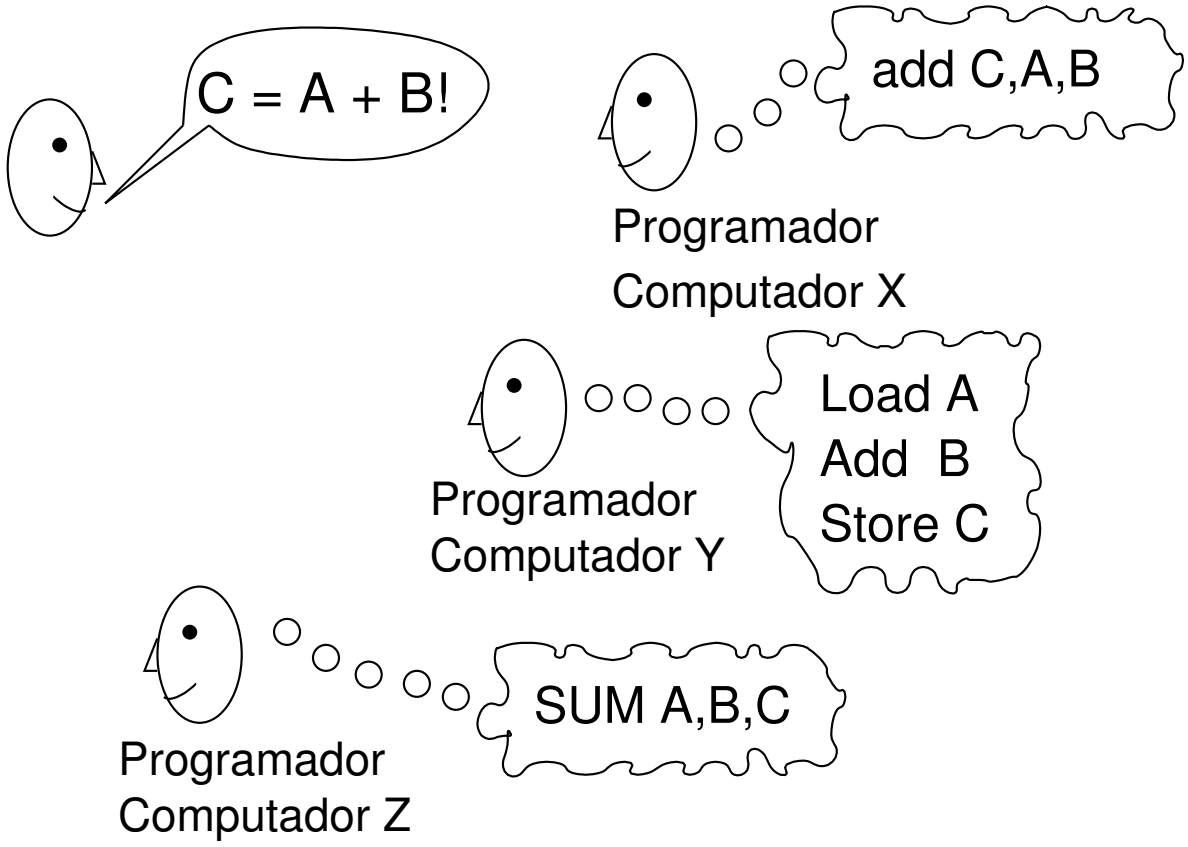
Ivan Ricarte

2008

Sumário

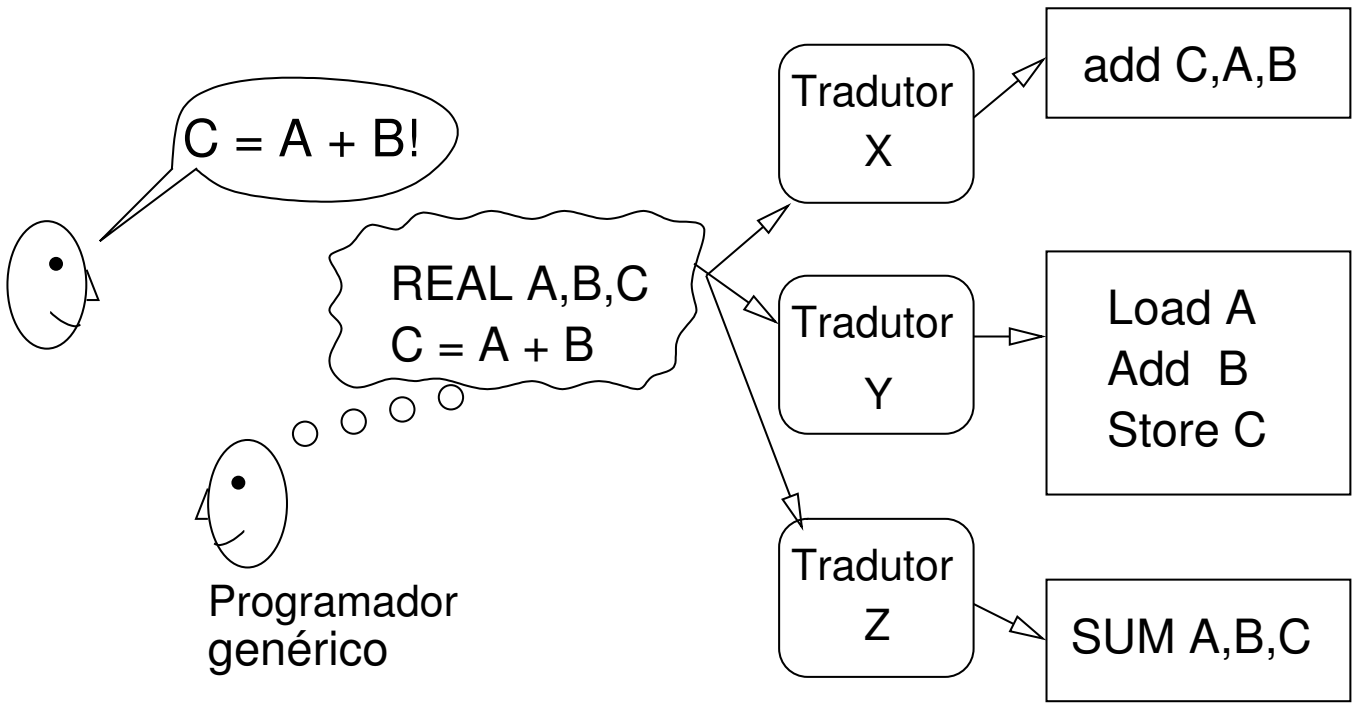
Para que compiladores?

A programação antes dos compiladores



Para que compiladores?

A programação com os compiladores



A linguagem dos processadores

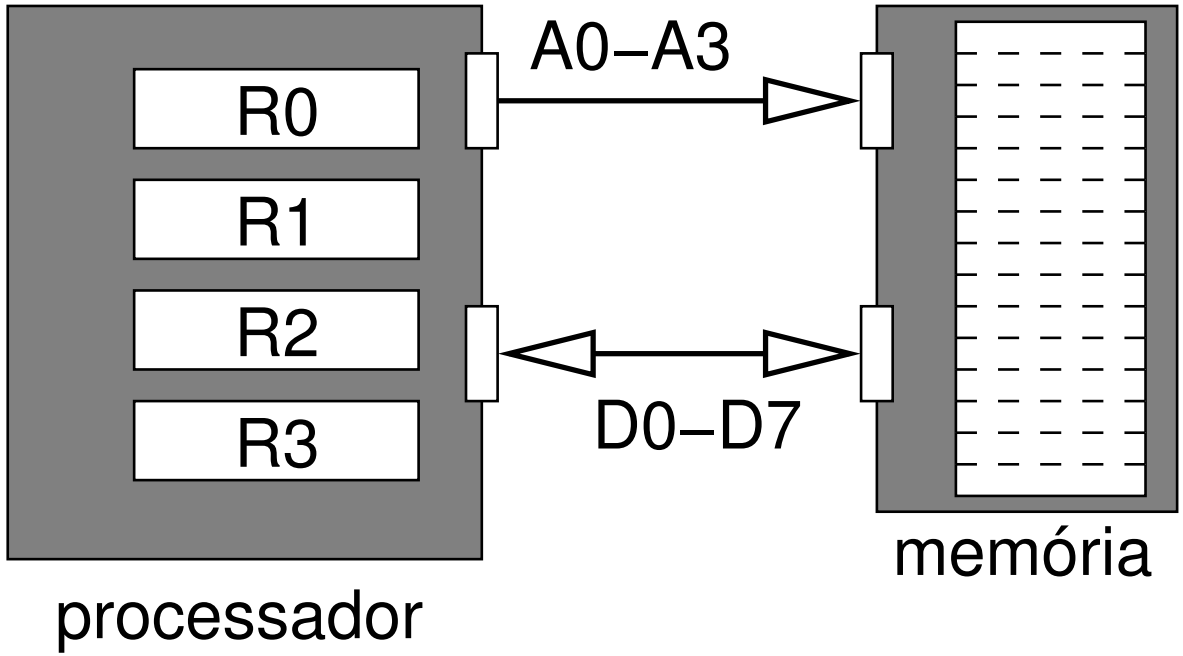
- Cada processador tem seu próprio jogo de instruções
  - Instruções refletem as características peculiares de cada processador — sua arquitetura
- Para cada instrução, duas representações

  Linguagem de máquina: seqüência de bits que codifica instrução no formato apropriado para a interpretação pelos circuitos

  Linguagem simbólica: representação mnemônica (textual)

Exemplo

Arquitetura de um processador hipotético



Exemplo

Instruções desse processador

LOAD	para transferir o conteúdo de uma posição de memória para um registrador.
STORE	para transferir o conteúdo de um registrador para uma posição de memória.
ADD	para adicionar o conteúdo de dois registradores e armazenar o resultado em outro registrador.
BZERO	para mudar o conteúdo do contador de programas para a posição de memória especificada se o conteúdo do registrador indicado for igual a zero.

Exemplo

Características do processador e impacto no formato da instrução de máquina

- Quatro instruções
  - $\Rightarrow$ Código de operação de dois bits:

LOAD	00
STORE	01
ADD	10
BZERO	11

- Quatro registradores de dados (além de outros não representados)
  - $\Rightarrow$ Endereços de registradores ocupam dois bits
- Capacidade de endereçar 16 posições de memória
  - $\Rightarrow$ endereços de memória ocupam quatro bits
- Palavras de oito bits
  - $\Rightarrow$ nenhum impacto

Exemplo

Um programa desse processador

Linguagem simbólica

```
LOAD  10, R1
LOAD  11, R2
ADD   R1, R2, R0
STORE R0, 12
```

Linguagem de máquina

```
00101001
00101110
10011000
01001100
```

Linguagens de alto nível

- Permitem expressar os comandos de forma independente do processador
- Comandos em linguagem mais próxima da humana
- Variações nas linguagens de acordo com seu objetivo

FORTRAN	Programação científica
C, C++	Programação de sistemas
Pascal	Programação estruturada
lisp	Processamento simbólico

Papel dos compiladores

Compilador

Programa que realiza a tradução de um programa em uma linguagem para um programa equivalente em outra linguagem

- Tipicamente: linguagem de alto nível $\longrightarrow$ linguagem de máquina
- mas também: linguagem de alto nível $\longrightarrow$ linguagem intermediária
- ou: linguagem intermediária $\longrightarrow$ linguagem de máquina
- ou ainda: linguagem de *script* $\longrightarrow$ linguagem de alto nível

Compiladores no processamento da informação

- Compiladores traduzem descrições de uma linguagem para outra
- Mesmo tipo de processamento está presente em aplicações que processam arquivos com informações
  - Processamento de arquivos XML
  - Processamento de páginas dinâmicas para a Web

Processamento de arquivos XML

XML

eXtensible Markup Language

- Aplicações em troca de informações entre processos distribuídos, em manutenção de arquivos de configuração e em representação de textos
- Arquivo organizado em elementos definidos por marcações e, eventualmente, conteúdos
- Existem regras básicas de formação para esses arquivos
- Esquemas podem restringir como esses elementos podem ser combinados

Exemplo de arquivo XML

```
<livro>
  <titulo>Dom Casmurro</titulo>
  <autor>Machado de Assis</autor>
  <ano>1900</ano>
</livro>
```

Como se compara o processamento desse arquivo à compilação de um programa em linguagem de alto nível?

Páginas dinâmicas na Web

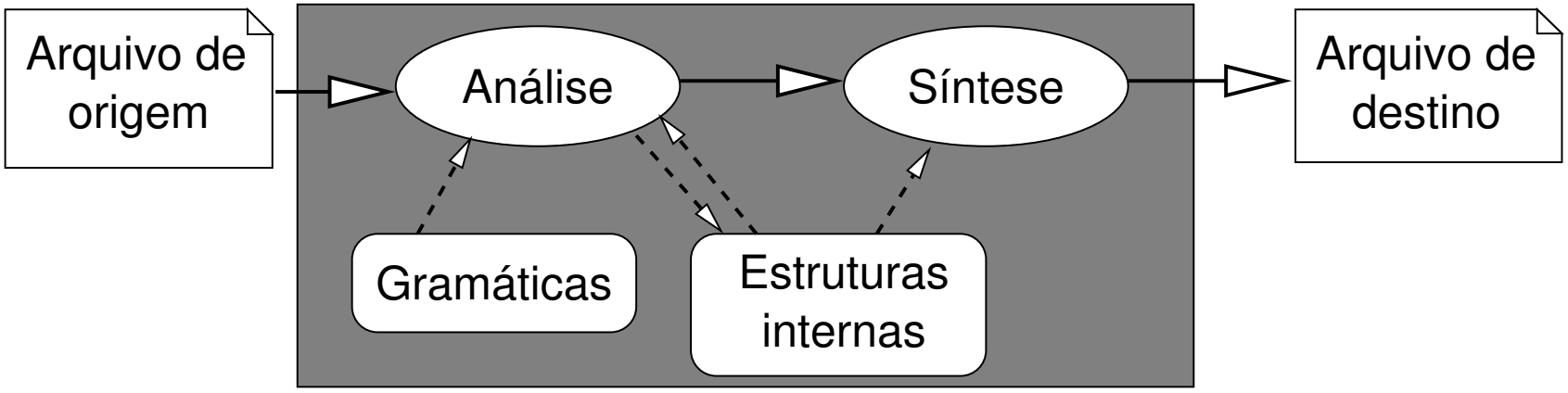
- Página dinâmica é criada no momento do atendimento a uma solicitação, a partir de uma descrição genérica que deve ser processada
- Tipicamente, descrição contém diretivas em uma linguagem de script que geram os conteúdos apresentados
- O resultado desse processamento é um arquivo com conteúdo em HTML (*Hypertext Markup Language*)

Exemplo de especificação de página dinâmica

```
<csp>
<@ #include ... %>
<% // código C ... %>
<html>
      Conteúdo em HTML
      com <%= código C %>
</html>
</csp>
```

Como se compara o processamento desse arquivo à compilação de um programa em linguagem de alto nível?

Atividades de um compilador



Análise

Análise léxica

Identificação dos elementos básicos da linguagem a partir dos caracteres individuais do arquivo fonte

Análise sintática

Reconhecimento, a partir dos elementos básicos, da estrutura de comandos básicos, de blocos de comandos, de funções, de programas

Síntese

Geração de código

Produção do código equivalente ao programa original a partir da estrutura construída na análise sintática

Otimização

Reorganização do código gerado para eliminar redundâncias e melhorar desempenho

Geração do código executável

- O compilador utiliza outros programas do sistema para produzir o código executável a partir do código em linguagem simbólica

  Montador Tradução do código em linguagem simbólica para o código de máquina

  Ligador Combinação de códigos de máquina de diferentes módulos num único módulo

Necessidade de manipulação de arquivos

- Entrada e saída para o compilador são arquivos

  Entrada arquivo de texto com a especificação de alto nível (código-fonte)

  Saída arquivo binário com o código executável em linguagem de máquina (módulo objeto)

- Outros arquivos temporários podem ser utilizados em etapas intermediárias

Arquivos em C++

- As necessidades de um compilador relativas à interação com arquivos restringem-se ao acesso seqüencial
- Para tal fim, C++ trabalha com o conceito de *streams*
  - Fontes ou destinos de dados acessados seqüencialmente
- Arquivos em disco podem ser manipulados como streams (fstream)

  ifstream para leitura seqüencial de arquivos
  ofstream para escrita seqüencial de arquivos

Leitura de arquivos

- Inicialmente, é preciso associar o arquivo ao objeto (variável) que será a ele associado

  Com o construtor

```
ifstream
nomeVariavel(nomeArquivo);
```

  Com o método open

```
ifstream nomeVariavel;
...
nomeVariavel.open(nomeArquivo);
```

Modo de abertura

- O modo padrão para abertura de um arquivo é considerar que seu conteúdo é texto
- Se conteúdo é binário, é preciso indicar esse fato na abertura do arquivo
- Especificar o *flag* `ios::binary` como segundo argumento do construtor ou do método open

Leitura do conteúdo do arquivo

Modo texto

- Para a leitura de caracteres de um arquivo texto (ifstream)

  Operador >> lê conteúdo com interpretação de formato, ignorando espaços em branco

  Método get lê cada caráter do arquivo sem interpretação, retornando o valor inteiro do código ASCII correspondente

  Método getline lê um bloco (uma linha) de caracteres para um arranjo de caracteres especificado pelo programador

Leitura do conteúdo do arquivo

Modo binário

- Para leitura de seqüências de bits de um arquivo

  Método read lê um bloco de bytes para um arranjo especificado pelo programador

eof

Qualquer que seja o tipo de conteúdo do arquivo, o método `eof` retorna `true` quando o final do arquivo é alcançado

Escrita de arquivos de destino

- Também é preciso associar o arquivo a um objeto

  Com o construtor

```
ofstream
nomeVariavel(nomeArquivo);
```

  Com o método open

```
ofstream nomeVariavel;
...
nomeVariavel.open(nomeArquivo);
```

Escrita do conteúdo em disco

- Para ofstream, outros especificadores além de `ios::binary` podem ser indicados na abertura do arquivo
- Caso arquivo já exista

ios::app	agrega novo conteúdo ao conteúdo já existente
ios::trunc	substitui conteúdo existente pelo novo conteúdo

Escrita do conteúdo em disco

- Operações para realizar a escrita no arquivo:

  Operador << formata (transforma valores de variáveis em caracteres)

  Método put escreve um caráter, sem interpretação

  Método write escreve um bloco de dados, sem interpretação

  Método flush descarrega o buffer interno de dados para o arquivo

Interação com arquivos padrão

- Todo programa ao executar inicia com três streams associados

cin	arquivo padrão para entrada de caracteres (ifstream)
cout	arquivo padrão para saída de caracteres (ofstream)
cerr	arquivo padrão para saída de mensagens de erro (ofstream)

- Para usá-los:

```
#include <iostream>
using namespace std;
```

Exemplo de compilador C++: gcc/g++

- Compilador C/C++ do Projeto Gnu, para ambientes do tipo unix
- Invocado sem outros argumentos a não ser o nome do arquivo, realiza quatro tarefas
  1. Pré-processamento (invoca programa cpp)
  2. Compilação
  3. Montagem (invoca programa as)
  4. Ligação (invoca programa ld)
- Resultado final é a criação de um arquivo executável no disco (a.out)
  - Nome padrão pode ser alterado com a chave -o

Processamento parcial

- Processamento pode ser interrompido após cada uma das etapas intermediárias por meio do uso de outras chaves para o programa
  1. Após pré-processamento: chave -E
     - Resultado é direcionado para a saída padrão
  2. Após compilação propriamente dita: chave -S
     - Resultado (arquivo texto, assembly) é criado num arquivo em disco com mesmo nome do arquivo de entrada mas com extensão .s
  3. Após montagem: chave -c
     - Resultado é criado num arquivo binário (código objeto) em disco com extensão .o

Indicação de erros de compilação

Considere a compilação do seguinte programa (hello.cpp):

```
#include <iostream>
using namespace std;
int main() {
  cout << "Oi, gente!" << endl;
}
```

Indicação de erros de compilação

Omissão de `;` na quarta linha:

```
hello.cpp: In function ‘int main()’:
hello.cpp:5: parse error before ‘}’ token
```

Por que o compilador gerou essa mensagem?

Indicação de erros de compilação

Omissão de " de abertura na quarta linha:

```
hello.cpp: In function ‘int main()’:
hello.cpp:4: ‘Oi’ undeclared (first use this function)
hello.cpp:4: (Each undeclared identifier is reported
     only once for each function it appears in.)
hello.cpp:4: ‘gente’ undeclared (first use this
     function)
hello.cpp:4: parse error before ‘!’ token
hello.cpp:4:21: warning: multi-line string literals
     are deprecated
hello.cpp:4:21: missing terminating " character
hello.cpp:4:21: possible start of unterminated string
     literal
```

Por que o compilador gerou essas mensagens?

Exemplo de processamento XML

- Analisador Xerces-C
  - Suíte de aplicativos para processamento de arquivos com conteúdo em XML
- Exemplo

  SAXCount se arquivo estiver correto, indica quantos elementos XML foram reconhecidos

Exemplo

Arquivo `livros.xml`:

```
<livros>
  <livro>
    <título>Dom Casmurro</título>
    <autor>Machado de Assis</autor>
    <ano>1900</ano>
  </livro>
  <livro>
    <título>As Intermitências da Morte</título>
    <autor>José Saramago</autor>
    <ano>2005</ano>
  </livro>
</livros>
```

Resultado:

livros.xml: 1 ms (9 elems, 0 attrs, 0 spaces, 118 chars)

Indicação de erros

Troca de primeiro `livro` por `lirvo`:

```
Fatal Error at file livros.xml, line 6, char 5
  Message: Expected end of tag 'lirvo'
```

Indicação de erros

Omissão da primeira marcação `/livro`:

```
Fatal Error at file livros.xml, line 11, char 8
  Message: Unterminated end tag, 'livro'
```

Indicação de erros

Omissão da marcação `/livros`:

```
Fatal Error at file livros.xml, line 13, char 1
  Message: The input ended before all started
           tags were ended.  Last tag started
           was 'livros'
```

Sugestões de leitura (Web)

- C++
  - Referência `http://www.cppreference.com/`
  - Compilador `http://gcc.gnu.org/`
- XML
  - Referência `http://www.w3.org/XML/`
  - Analisador `http://xerces.apache.org/xerces-c/`