

Noções de compilação

Compilador: o que é, para que serve e estrutura geral

Parentes do compilador e programas correlatos

Prof. Thiago A. S. Pardo

1

Exercício em duplas

- Para esquentar...



2

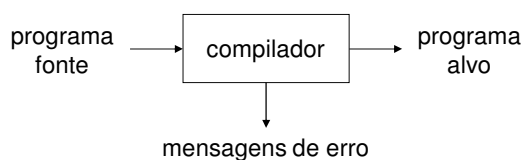
Compilação: por que estudar? (parte 1)

- Compiladores: uma das **principais ferramentas** do cientista/engenheiro da computação
- Técnicas de compilação se aplicam a **projetos gerais de programas**
 - Editores de texto, sistemas de recuperação de informação, reconhecimento de padrões, processamento de línguas
 - Composição tipográfica e desenho de figuras
 - Teste de programas
- Utilização de conceitos e métodos de **diversas disciplinas**
 - Algoritmos
 - Linguagens de programação
 - Teoria das linguagens
 - Engenharia de software
 - Arquitetura de computadores

3

Compilador: o que é e para que serve

- **Definição:** Programa que lê um programa em uma linguagem-fonte e o traduz em um programa em uma linguagem-alvo (objeto)
 - Linguagem-fonte: Pascal, C
 - Linguagem-alvo: linguagem de montagem (*assembly*), código de máquina
- Durante o processo de tradução, relatam-se erros encontrados



4

Um pouco de história

- Inicialmente, programação em código de máquina
- Programação em linguagem de montagem
 - Maior “facilidade de programação”
 - Necessidade de um montador

- Não há magia!

```

ORG 0H      ;start (origin) at location 0
MOV R5,#25H ;load 25H into R5
MOV R7,#34H ;load 34H into R7
MOV A,#0    ;load 0 into A
ADD A,R5    ;add contents of R5 to A
            ;now A = A + R5
ADD A,R7    ;add contents of R7 to A
            ;now A = A + R7
ADD A,#12H  ;add to A value 12H
            ;now A = A + 12H
HERE:SJMP  HERE ;stay in this loop
END         ;end of asm source file

```

- Finalmente, linguagens de mais alto nível

Um pouco de história

- Primeiros compiladores começaram a surgir no início dos anos 50
 - Diversos experimentos e implementações realizados independentemente
 - Trabalhos iniciais: tradução de fórmulas aritméticas em código de máquina
 - Compiladores eram considerados programas muito difíceis de construir
 - Primeiro compilador Fortran levou 18 homens-ano para ser construído

Um pouco de história

- Desde então, **técnicas sistemáticas** para construção de compiladores foram identificadas
 - Reconhecimento de cadeias, gramáticas, geração de linguagem
- Desenvolvimento de **boas linguagens** e ambientes de programação
 - C, C++, bibliotecas, linguagens visuais
- Desenvolvimento de **programas para produção automática** de compiladores
 - lex, yacc
- **Atualmente, um aluno de graduação pode construir um compilador rapidamente**
 - Ainda assim, programa bastante complexo
 - Estimativa de código de 10.000 a 1.000.000 de linhas

7

Um pouco de história

- Antes um mistério, agora uma das áreas mais conhecidas
 - **1957**: Fortran – primeiros compiladores para processamento de expressões aritméticas e fórmulas
 - **1960**: Algol – primeira definição formal de linguagem, com gramática na forma normal de Backus, estruturas de blocos, recursão, etc.
 - **1970**: Pascal – tipos definidos pelos usuários, máquina virtual (P-Code)
 - **1985**: C++ – orientação a objetos, exceções
 - **1995**: Java – compilação *just-in-time* (traduz bytecodes para código de máquina e executa), melhorando o tempo e execução do programa, portabilidade

8

Um pouco de história

- Há quem **odeie** C, C++, Java, etc.
 - Muito detalhe, baixo nível
 - Tendência para programação orientada a componentes, linguagens visuais, linguagens de altíssimo nível (por exemplo, **Haskell**), frameworks
 - Versão “computeira” da discussão do carro “manual” vs. “automático”

9

Compilação

- **Exigências e características atuais**
 - Gere código corretamente
 - Seja capaz de tratar de programas de qualquer tamanho
 - Velocidade da compilação não é a característica principal
 - Tamanho do compilador já não é mais um problema
 - *User-friendliness* se mede pela qualidade das mensagens de erros
 - A importância da velocidade e tamanho do código gerado depende do propósito do compilador - velocidade vem em primeiro lugar, normalmente

10

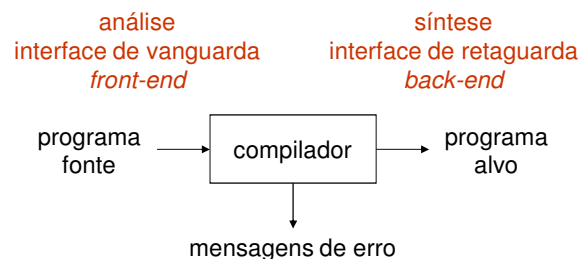
Compilação

- Em geral, somente **linguagens imperativas** são estudadas em cursos de Compilação
 - Linguagens funcionais (LISP) e lógicas (Prolog) requerem técnicas diferentes
 - Por quê?

11

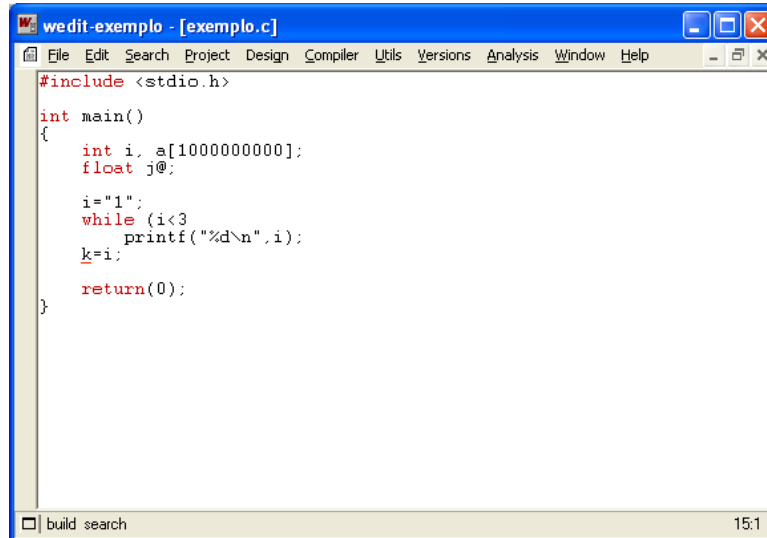
Modelo de compilação

- Duas etapas
 - Análise: interpreta o programa-fonte e cria uma representação intermediária do mesmo
 - Síntese: a partir da representação intermediária, produz o programa-alvo



12

Exemplo



```

wedit-exemplo - [exemplo.c]
File Edit Search Project Design Compiler Utils Versions Analysis Window Help
#include <stdio.h>

int main()
{
    int i, a[1000000000];
    float j@;

    i="1";
    while (i<3
        printf("%d\n",i);
    k=i;

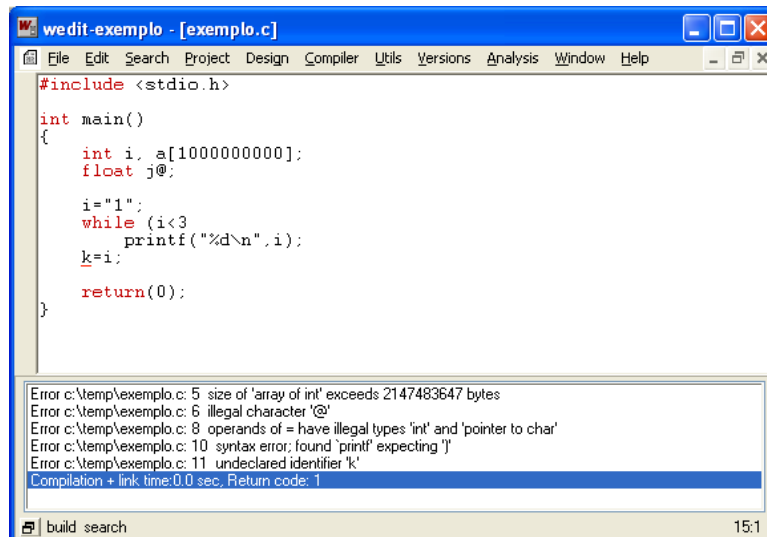
    return(0);
}

build search 15:1
  
```

Quais os erros?

13

Exemplo



```

wedit-exemplo - [exemplo.c]
File Edit Search Project Design Compiler Utils Versions Analysis Window Help
#include <stdio.h>

int main()
{
    int i, a[1000000000];
    float j@;

    i="1";
    while (i<3
        printf("%d\n",i);
    k=i;

    return(0);
}

Error c:\temp\exemplo.c: 5 size of 'array of int' exceeds 2147483647 bytes
Error c:\temp\exemplo.c: 6 illegal character '@'
Error c:\temp\exemplo.c: 8 operands of = have illegal types 'int' and 'pointer to char'
Error c:\temp\exemplo.c: 10 syntax error: found 'printf' expecting ')'
Error c:\temp\exemplo.c: 11 undeclared identifier 'k'
Compilation + link time:0.0 sec. Return code: 1

build search 15:1
  
```

Quais os tipos dos erros?

14

Exemplo

```

#include <stdio.h>

int main()
{
    int i, a[1000000000];
    float j@;

    i="1";
    while (i<3
        printf("%d\n",i);
    k=i;
}
  
```

Error c:\temp\exemplo.c: 5 size of 'array of int' exceeds 2147483647 bytes
 Error c:\temp\exemplo.c: 6 illegal character '@'
 Error c:\temp\exemplo.c: 8 operands of = have illegal types 'int' and 'pointer to char'
 Error c:\temp\exemplo.c: 10 syntax error; found 'printf' expecting ')'

O que diferencia os tipos de erros?

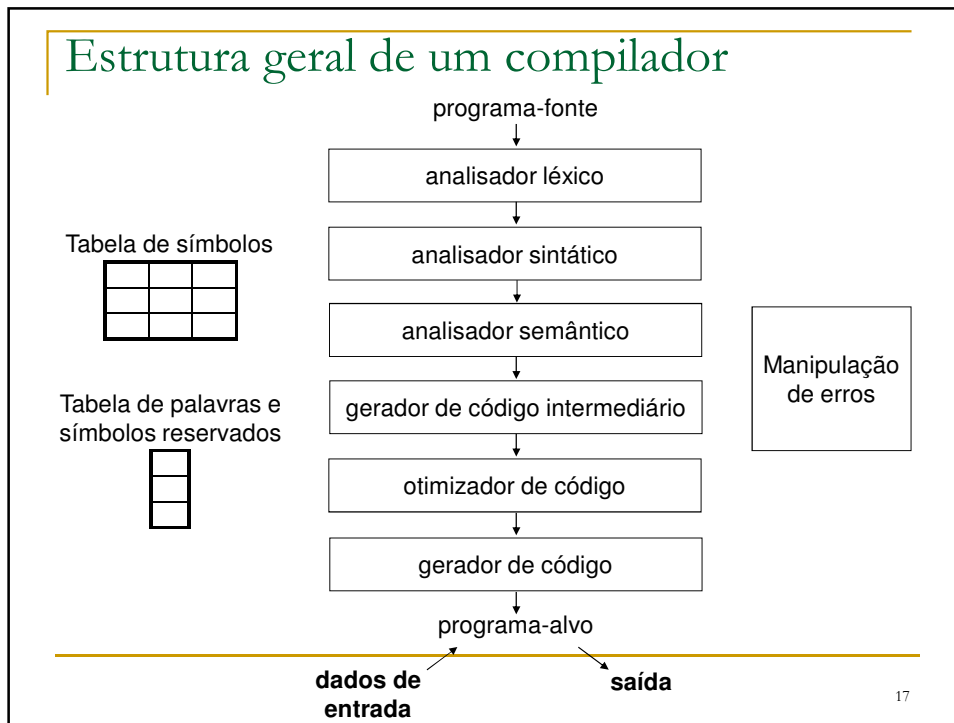
15

Fases da compilação

- Lexical: palavras (*tokens*) do programa
 - i, while, =, [, (, <, int
 - Erro: **j@**
- Sintática: combinação de tokens que formam o programa
 - comando_while → while (expressão) comandos
 - Erro: **while (expressão comandos**
- Semântica e contextual: adequação do uso
 - Tipos semelhantes em comandos (atribuição, por exemplo), uso de identificadores declarados
 - Erros: **i="1", k=i**
- Geração de código: especificidades da máquina-alvo e sua linguagem
 - Alocação de memória, uso de registradores
 - Erro: **a[1000000000]**

16

Estrutura geral de um compilador



Estruturas da compilação

- Como diferenciar palavras e símbolos reservados (while, int, :=) de identificadores definidos pelo usuário
 - Tabela de palavras e símbolos reservados

int
while
:=
...

Estruturas da compilação

- Como saber durante a compilação de um programa o tipo e o valor dos identificadores, escopo das variáveis, número e tipo dos parâmetros de um procedimento, etc.
 - Tabela de símbolos

Identificador	Classe	Tipo	Valor	...
i	var	integer	1	...
fat	proc	-	-	...
...				

19

Analizador lexical

- Reconhecimento e classificação dos tokens
 - Expressões regulares, autômatos

$x := x + y * 2$



$\langle x, id_1 \rangle \langle :=, := \rangle \langle x, id_1 \rangle \langle +, op \rangle \langle y, id_2 \rangle \langle *, op \rangle \langle 2, num \rangle$

20

Analizador sintático

- Verificação da formação do programa
 - Gramáticas livres de contexto

$\langle x, id_1 \rangle \langle :=, := \rangle \langle x, id_1 \rangle \langle +, op \rangle \langle y, id_2 \rangle \langle *, op \rangle \langle 2, num \rangle$



comando_atribuição $\rightarrow id_1 := id_1 \text{ op } id_2 \text{ op num}$

21

Analizador semântico

- Verificação do uso adequado

$id_1 := id_1 \text{ op } id_2 \text{ op num}$



$(id_1)_{int} := (id_1 \text{ op } id_2 \text{ op num})_{int}$
 busca_tabela_símbolos(id_1)=TRUE
 busca_tabela_símbolos(id_2)=TRUE

22

Gerador de código intermediário

- Geração de código intermediário/preliminar

$id_1 := id_1 \text{ op } id_2 \text{ op num}$



temp1 := $id_2 * 2$
temp2 := $id_1 + temp1$
 $id_1 := temp2$

$x := x + y * 2$

23

Otimizador de código

- Otimização do código intermediário

temp1 := $id_2 * 2$
temp2 := $id_1 + temp1$
 $id_1 := temp2$



temp1 := $id_2 * 2$
 $id_1 := id_1 + temp1$

$x := x + y * 2$

24

Gerador de código

- Geração do código para a máquina-alvo

```
temp1 := id2 * 2  
id1 := id1 + temp1
```



```
MOV id2 R1  
MULT 2 R1  
MOV id1 R2  
ADD R1 R2  
MOV R2 id1
```

25

Exemplo: compilação passo a passo

```
program p1;  
var x: integer;  
begin  
  read(x);  
  x:=x*2;  
  write(x);  
end.
```

26

Definição de linguagens de programação

- **Definição de uma linguagem**, ou manual de referência da linguagem
 - Em geral, estruturas léxica e sintática são especificadas formalmente
 - A semântica muitas vezes é descrita em língua natural

- **Padrões internacionais**
 - ANSI – *American National Standards Institute*
 - ISO – *International Organization for Standardization*

27

Passagens

- Passagem: leitura de um arquivo de entrada e escrita de um arquivo de saída

- Compiladores podem ter **várias passagens**
 - Esquema anterior de compilação
 - código-fonte → código intermediário → código-alvo
 - Maior tempo de leitura e escrita
 - Útil quando há pouca memória disponível, quando a linguagem é complexa ou quando se visa portabilidade

- Compilador de uma única passagem
 - código-fonte → código-alvo
 - Todo processo de compilação em memória: dados de fases diferentes podem ser necessários para a compilação

28

Compilação em uma passagem

- Em geral, em compiladores de uma passagem, o **analisador sintático é o “chefe”**
 - Gerencia todo o processo, ativando as etapas anteriores e posteriores
 - Código é gerado na medida em que o programa é interpretado
 - Processamento *interleaved* (vs. *pipeline*)

29

Sistemas correlatos

- **Interpretadores**: executam diretamente instrução por instrução do código-fonte
- **Processadores de macros** (por exemplo, *defines* em C)
- **Montadores (*assemblers*)**: traduzem linguagem de montagem em linguagem de máquina
- **Carregadores**: alocação de instruções de programação e dados na memória
- **Editores de ligação**: criação de um único programa a partir de diversos programas compilados
- **Pré-processadores**: retiram comentários, podem processar macros, etc.
- **Editores/IDE baseados em estrutura**: indicam erros durante edição do programa, fazem *code completion*
- **Depuradores**
- Etc.

30

Interpretadores vs. compiladores

- **Interpretadores**
 - Menores que os compiladores
 - Mais adaptáveis a ambientes computacionais diversos
 - Melhor diagnóstico de erro (interpretação linha a linha)
 - Tempo de execução maior
 - Instruções de um loop são analisadas e executadas N vezes!
 - Javascript, Python, Perl
- **Compiladores**
 - Compila-se uma única vez, executando-se quantas vezes se queira
 - Tempo de execução menor
 - C, Pascal
- **Linguagens híbridas**
 - Java: compilada para um código intermediário/virtual (*bytecodes*), que, por sua vez, é interpretado (*virtual machine*)
 - .NET: compilada para o código intermediário *Microsoft Intermediate Language* (MSIL), interpretado pela máquina virtual *Common Language Runtime* (CLR)

31

Classificação de compiladores

- Classificam-se compiladores em função de vários fatores
 - Código que gera
 - Para quem gera código
 - Ambiente de execução
 - Etc.

32

Classificação de compiladores

■ Código que gera

- Linguagem de máquina pura
- Linguagem de máquina aumentada com rotinas do sistema operacional (acesso a BIOS, registradores, I/O)
 - Abordagem mais comum
- Linguagem de montagem
- Linguagem de máquina virtual, em que as instruções são completamente virtuais e necessitam de posterior interpretação
 - Exemplo?

■ Código absoluto vs. relocável

33

Classificação de compiladores

■ Para quem gera código

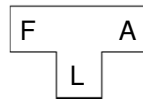
- Compilador auto-residente: executado na mesma máquina para a qual gerou código
 - Traduz L para a máquina M, executando na máquina M
- Compilador cruzado: roda em uma máquina e produz código para outra
 - Traduz L para a máquina M, mas executa na máquina R
- Compilador auto-compilável: compilador para uma linguagem X que é implementado na própria linguagem X
 - Traduz L para a máquina M, e é implementado em L

34

Classificação de compiladores

■ Diagramas-T

- Compilação de uma linguagem-fonte F para uma linguagem-alvo A, com um compilador implementado na linguagem L

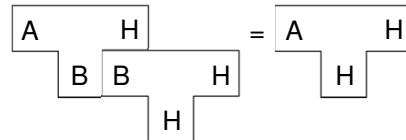
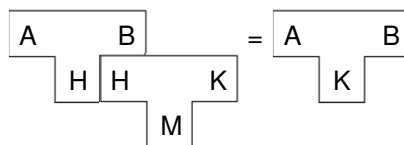
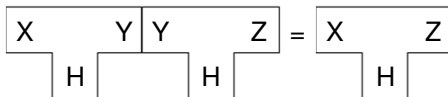


35

Classificação de compiladores

■ Diagramas-T

- Úteis para esquematização dos compiladores



36

Ferramentas para compilação

- *Compiler compilers*

- Lex, Flex
- Yacc, Bison, JavaCC
- Muitos outros

37

Compilação: por que estudar? (parte 2)

- **Várias aplicações e necessidades atuais**

- Validação de arquiteturas diferenciadas de computadores
- Aceitação de novas linguagens de programação
- Otimização de código: celulares, sistemas embarcados, novas arquiteturas
- Teste de falhas/erros em software
- Busca for brechas/falhas de segurança em sistemas
- Efetividade de paralelismo (ambientes *multicore*)
- Melhor uso de memória (registradores, caches, memória RAM)
- Tradução entre sistemas diferentes, síntese de hardware (da especificação para o modelo)
- Interpretação de linguagens especiais: SQL, por exemplo
- Etc.

38

Exercício – temas para pesquisar

■ Para pesquisar em casa

- Quais as características de uma linguagem que determinam que ela deve ser compilada ou interpretada?
- O que é decompilação e quais seus passos básicos?
- Como é o mercado de compiladores no Brasil?