Linguagem C
Introdução

Contexto Histórico

Principais diferenças do Java

Funções em C

Compilar programas em C no Linux

Porquê C em AC?

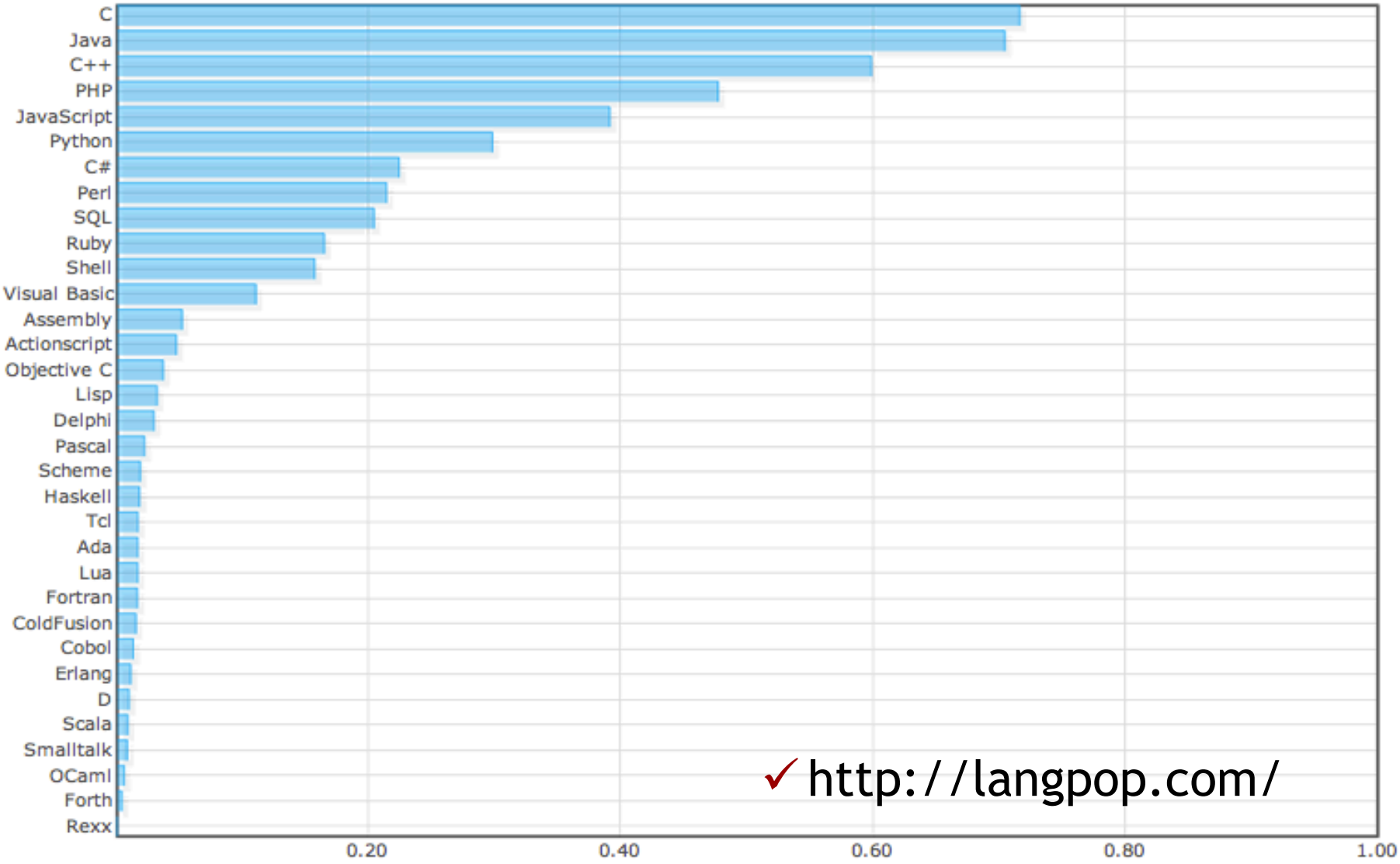
- A linguagem C fornece um modelo de programação próximo da máquina física
  - ✓ Que vamos estudar nesta cadeira
- Não requer uma máquina virtual
- Permite a manipulação directa de endereços de memória (apontadores)
- Considerada por alguns autores como um "assembly portável"
- É relativamente fácil ligar programas em C com módulos em assembly

Mas afinal alguém usa C?

- O C está entre as linguagens mais usadas e mais populares
  - ✓ http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Position Feb 2011	Position Feb 2010	Delta in Position	Programming Language	Ratings Feb 2011	Delta Feb 2010	Status
1	1	=	Java	18.482%	+1.13%	A
2	2	=	C	14.986%	-1.62%	A
3	4	↑	C++	8.187%	-1.26%	A
4	7	↑↑↑	Python	7.038%	+2.72%	A
5	3	↓↓	PHP	6.973%	-3.03%	A
6	6	=	C#	6.809%	+1.79%	A
7	5	↓↓	(Visual) Basic	4.924%	-2.13%	A
8	12	↑↑↑↑	Objective-C	2.571%	+0.79%	A
9	10	↑	JavaScript	2.558%	-0.08%	A
10	8	↓↓	Perl	1.907%	-1.69%	A
11	11	=	Ruby	1.615%	-0.82%	A
12	-	=	Assembly*	1.269%	-	A-
13	9	↓↓↓↓	Delphi	1.060%	-1.60%	A
14	19	↑↑↑↑↑	Lisp	0.956%	+0.39%	A
15	37	↑↑↑↑↑↑↑↑↑↑	NXT-G	0.849%	+0.58%	A–
16	30	↑↑↑↑↑↑↑↑↑↑	Ada	0.805%	+0.44%	A–
17	17	=	Pascal	0.735%	+0.13%	A
18	21	↑↑↑	Lua	0.714%	+0.21%	A–
19	13	↓↓↓↓↓↓	Go	0.707%	-1.07%	A–
20	32	↑↑↑↑↑↑↑↑↑↑	RPG (OS/400)	0.626%	+0.27%	A–

Mas afinal alguém usa C?



✓ http://langpop.com/

Linguagem C - origem

- Desenvolvida por Dennis Ritchie nos anos 70
- Chamada C por derivar de uma linguagem anterior chamada B (que derivava do BCPL)
- Objectivo — desenvolver sistemas de operação e compiladores
  - ✓ Hoje também muito usada para implementar máquinas virtuais de outras linguagens
- No entanto é uma linguagem de uso geral
- Primeiro êxito — reescrita do sistema UNIX em C a partir da versão original em assembly

Linguagem C - evolução

- 1978 – Kernighan & Ritchie publicam a primeira edição de "*The C Programming Language*" efectivamente definindo a linguagem
- 1983 – ANSI C – é definido um standard internacional para a linguagem C. Mais tarde é publicada a segunda edição do livro, cobrindo este standard
- 1999 – C99 – O standard é refinado, incluindo algumas inovações que foram introduzidas pelo C++

Linguagem C – sucessores famosos

- C++
  - ✓ Entre 1979 e 1983 Bjarne Stroustrup desenvolve uma extensão à linguagem C
  - ✓ Orientada aos objectos — classes
  - ✓ Inclui excepções — tratamento de erros sofisticado
  - ✓ Sistema de tipos mais forte que o do C
- Java
  - ✓ Em 1996 a Sun lança a primeira implementação da linguagem Java
  - ✓ Inovações principais do Java em relação ao C++:
    - ✓ Gestão de memória automática
    - ✓ Máquina virtual standard — a JVM

Principais diferenças entre C e Java

- C não tem classes
  - ✓ É baseado em funções
- C não tem excepções
  - ✓ Tratamento de erros a cargo do programador
- C permite manipular directamente endereços de memória (apontadores)
  - ✓ A maior fonte de erros de programas em C
- C não tem gestão de memória automática
  - ✓ Não tem *garbage collection*
- O sistema de tipos do C é fraco
  - ✓ O compilador de C é muito mais permissivo que o de Java

Primeiro programa em C

```
// usar a biblioteca de I/O
#include <stdio.h>

// função principal
int main()
{     /* escrever a mensagem no ecrã
         e mudar de linha */
      printf("hello world\n");

      return 0; // retornar 0 ao sistema operativo
}
```

C e Java – Classes vs funções

- Em Java um programa é um conjunto de classes
- Em C um programa é um conjunto de **funções**
- Uma função em C corresponde, grosso modo, a um método em Java!
- Exemplo de uma função em C:

```
int quadrado( int x )
{
  return x*x;
}
```

C e Java – tipos de dados básicos

- Essencialmente os mesmos, no entanto:
- Em C não existe **boolean**
  - ✓ temos que usar inteiros: 0 significa falso; qualquer outro valor significa verdade
- **char** em C corresponde a **byte** em Java
  - ✓ Os caracteres correspondem aos seus códigos ASCII
  - ✓ Por exemplo: '0' = 48, 'A' = 65, '\n' = 10
  - ✓ Logo podemos misturar livremente caracteres e inteiros em expressões e atribuir inteiros a caracteres e vice-versa
- **Cuidado**! String não é um tipo básico em Java mas sim uma classe (e não existe em C...)

Função main

- É a função “principal” de um programa
- Todos os programas têm que ter uma função `main`
- Para todos os efeitos é uma função como qualquer outra, mas que é chamada automaticamente quando o programa começa
- Retorna um inteiro ao sistema operativo (convenciona-se que 0 indica sucesso e outro valor será um código de erro)

Protótipos das funções

- Em C uma função pode ter um protótipo que indica ao compilador como ela deve ser chamada
- Corresponde à assinatura do método em Java
- Tem de corresponder exactamente à declaracão da função
- O protótipo é importante quando a função é chamada antes de ser definida

```
//protótipo da função main (sem argumentos)
int main(void);
// protótipo da função quadrado
int quadrado(int);
```

C - I/O – printf e scanf

- As funções de I/O do C não fazem parte da linguagem mas da biblioteca standard!
- São funções como todas as outras!
- Os seus protótipos estão declarados no ficheiro do sistema `stdio.h`

```
#include <stdio.h>

printf("formatação", a1, a2 ... );
scanf("formatação", &a1, &a2 ...);
```

C - I/O - printf e scanf

- Caracteres de formatação:
  - ✓ `%c` – caracter
  - ✓ `%d` ou `%i` – inteiro decimal
  - ✓ `%x` – inteiro hexadecimal
  - ✓ `%s` – string (vector de caracteres)
  - ✓ `%f` – float
  - ✓ `%lf` – double
- Exemplos:

```
printf("nome %s\n idade %d\n", nome, idade);
printf("decimal  %d = hex %x\n", n, n);
scanf("%d", &idade);
scanf("%s",nome);
```

Segundo programa em C

```
#include <stdio.h>
int quadrado( int x )
{     return x*x;
}
int main()
{     int n;              // variável local
      printf("Introduza n:");
      scanf("%d",&n);
      printf("O quadrado de %d é %d\n",
                        n, quadrado(n) );
      return 0;
}
```

C - compilar

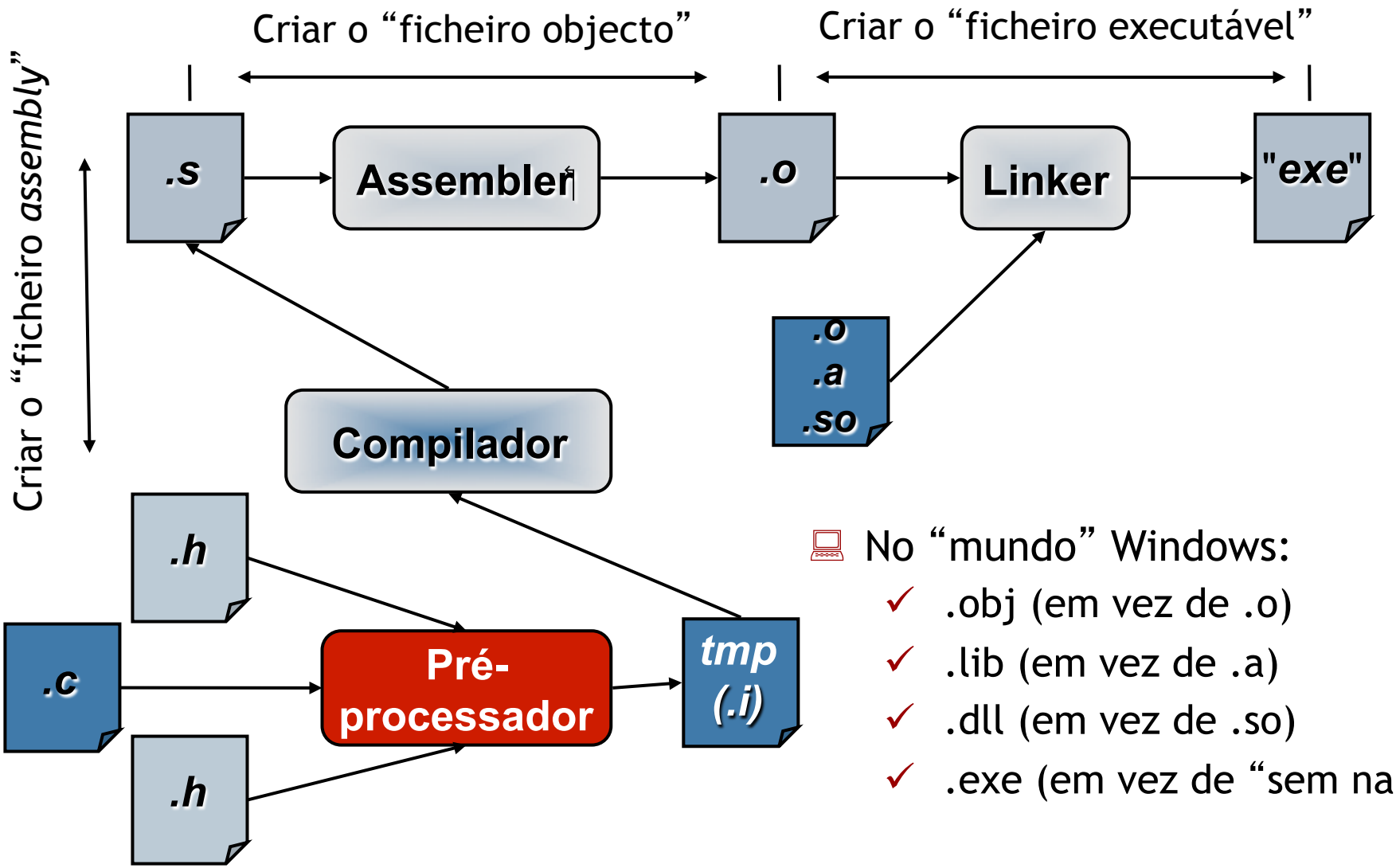
- Linha de comandos:

```
gcc -Wall -g -o prog prog.c
```

  - ✓ **gcc**: compilador
  - ✓ **-Wall**: dar todos os avisos (**importante!**)
  - ✓ **-g**: preparar para debugger
  - ✓ **-o prog**: especifica o nome do executável
  - ✓ **prog.c**: programa fonte em C

- Correr o programa
  - ✓ `./prog`
- Manual (linha de comandos):
  - ✓ `man gcc`
  - ✓ `man 3 printf` *(ou outra função da biblioteca)*

Compilar um programa C em Linux



- No "mundo" Windows:
  - ✓ .obj (em vez de .o)
  - ✓ .lib (em vez de .a)
  - ✓ .dll (em vez de .so)
  - ✓ .exe (em vez de "sem nada")

Pré-processador — Directivas

- Fase de substituição de texto executada antes da compilação
- Usa directivas em linhas iniciadas por #
- **#include** – incluir outros ficheiros no programa
  - ✓ Por exemplo, cabeçalhos de bibliotecas:

```
#include <string.h>
#include <stdlib.h>
```

- **#define** – substituição de texto
  - ✓ Muito usada para definir constantes

```
#define TRUE 1
#define FALSE 0
```

Programa antes de pré-processado

```
#include <stdio.h>
// colocar sempre números e expressões entre parêntesis
#define PI (3.1416)

float graus_para_rad(int x) {
  return x*PI/180;
}

int main() {
  int x = 10;
  float res = graus_para_rad(x);
  printf("O valor em radianos de %d e' %f\n", x, res);
  return 0;
}
```

Programa depois de pré-processado

```
...
int printf(const char format, ...);
...

float graus_para_rad(int x) {
  return x*(3.1416)/180;
}

int main() {
  int x = 10;
  float res = graus_para_rad(x);
  printf("O valor em radianos de %d e' %f\n", x, res);
  return 0;
}
```

Código do ficheiro stdio.h.

Note que não contém a implementação do printf. Só o seu protótipo

Susbtituição de PI pelo valor definido