C x Java

- Não possui suporte a orentação a objetos
- Linguagem de nível intermediário:
  - controle mais direto do hardware, porém
  - também suporta estruturas complexas
- Gerenciamento de memória explícito
- Detecção de erro explícita (sem try/catch)
- Maior performance do programa final
- Maior dificuldade de manutenção

Linguagem C – Exemplo 1

```c
// exemplo01.c
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

Escreve “Hello World!” (sem aspas) na saida padrão (no caso: um terminal no Linux ou um prompt de comando no Windows)

Exemplo 1 – Compilação (gcc)

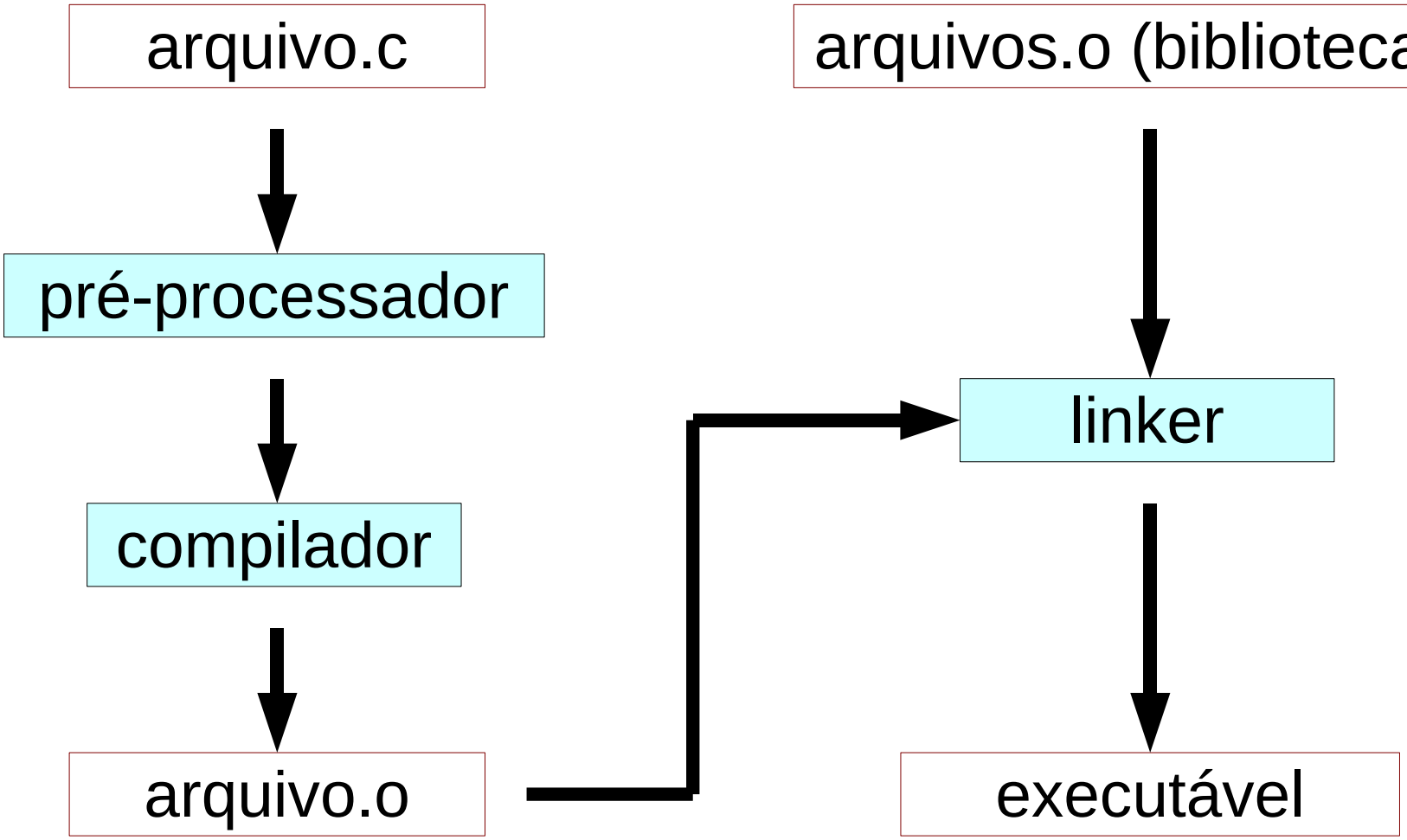
```
/* exemplo01.c */
#include <stdio.h>

int main() {
   printf("Hello World!\n");
   return 0;
}
```

Em um terminal shell:

```
$ gcc exemplo01.c -o hello
$ ./hello
Hello World!
$
```

Linguagem C – Compilação



Linguagem C – Tipos primitivos

- C x Java:
  - char tem 8 bits (não é 16 bits como em Java)
  - não existe tipo booleano (usar int ou char):
    - = 0 falso
    - ≠ 0 verdadeiro
  - tipos inteiros podem ser signed ou unsigned
  - não tem tipo *string*: usa-se vetor de char

Linguagem C – Tipos primitivos

Numa máquina 64 bits, os tipos de dados primitivos são:			
Nome	# bytes	signed	unsigned
char	1	-2^07 a (2^07 - 1)	0 a 255
short	2	-2^15 a (2^15 - 1)	0 a 65.535
int	4	-2^31 a (2^31 - 1)	0 a 4.294.967.295
long	8	-2^63 a (2^63 - 1)	0 a 18.446.744.073.709.551.615
float	4	7 dígitos de precisão	-
double	8	15 digitos de precisão	-

- Todos os tipos inteiros são *signed* por default
- Número de bytes depende da máquina
- Em caso de dúvida use `sizeof(tipo)`: retorna número de bytes que o tipo ocupa na memória

Linguagem C – Literais

- int: 1234 ou 02322 (octal) ou 0x4D2 (hexa.)
- long: 60238097324 ou sufixo L; exemplo:
  - 1234L ou 02322L ou 0x4D2L
- double: 123.4 ou 1234.0 ou 12e-2 (= 0.12)
- char: literal inteiro entre -128 e 127 ou um caractere entre aspas simples:
  - 'a' (= 97 em ASCII)
  - '0' (= 48 em ASCII)
  - etc...

Linguagem C – Literais (char)

Caractéres especiais (escape characters):

```
\n – newline
\t – tab
\r – carriage return
\b – backspace
\a – bell
\\ – backslash
\" – double quote
\0 – null (tem valor 0)
```

C – Declarações de Variáveis

```
tipo nome1, nome2, ...;
tipo n1 = v1, n2 = v2, ...;
```

- Exemplos:

```
int i, j, k;
long a, b = 0x2322L;
double x, y = 2.36, z;
char c = 'a', d = '\n';
```

- Declarar variáveis antes de usá-las
- Variáveis não inicializadas contêm “lixo”

C – Declaração de Variáveis

- Nome da variável pode conter os caractéres:
  - A-Z, a-z, 0-9, _ (underscore)
  - não pode começar com número
  - não pode ser uma palavra chave de C
  - nomes de funções seguem a mesma regra

- Exemplos:

```
int senha_fraca = 17, usuario1;
int usuario2;
```

Linguagem C – Conversão de tipo

- Exemplo 1:

```
int a = 10; long b;
b = (long) a;
```

- Valor preservado, só o tipo é alterado
- Exemplo 2:

```
int a; long b = 23;
a = (int) b;
```

- Perigoso: se `b` não “cabe” num `int`, seu valor será truncado

Linguagem C – Conversão de tipo

```
int a = 17, b = 3;
double x = a / b;
```

- Neste caso `x` contém o valor `5`

```
int a = 17, b = 3;
double x = a / (double) b;
```

- Neste caso `x` contém o valor `5.66667`

```
double x = 5.93487;
int j = (int) x;
```

- Neste caso `j` contém o valor `5`

Linguagem C – Vetores

- Para declarar um vetor de inteiros de comprimento 100 faça:

```
int v[100];
```

- Note que os índices vão de `0` a `99`
- Para inicializar o vetor com zeros faça:

```
int v[100] = {0};
```

- Para inicializar com outra constante faça manualmente (usando um laço `for`)
- O gcc aceita a sintaxe

```
int v[100] = {[0 ... 99] = 5};
```

Linguagem C – Vetores

- Literais de vetores só podem ser usados na ininicialização de um vetor recém-declarado
- O código abaixo é válido:

```
int v[7] = {6, 34, 8, 265, 4, 8, 3};
```

- O código abaixo não é válido:

```
int v[7];
v[7] = {6, 34, 8, 265, 4, 8, 3};
v[7] = {0,};
```

Linguagem C – *Strings*

- São vetores de char terminados por `'\0'`

```
char nome[4] = {'a', 'b', 'a', '\0'};
```

- Também é possível deixar que o compilador determine o tamanho do vetor ao fazer:

```
char nome[] = "Estrutura de dados";
```

- O vetor acima tem 19 posições: 18 para os caracteres e mais uma que contém o caractere nulo `'\0'` para indicar o fim da string.
- Funções de manipulação de strings são declaradas em `<string.h>`

Linguagem C – Funções

- Como em java:
  - podem receber parâmetros
  - podem devolver um valor
  - podem ser recursivas
- Devem ser declaradas antes de serem usadas
- Declaração de função (sem sua definição):

```
double divide(double, double);
long fatorial(int);
int lg(long);
```

Linguagem C – Funções

- Definição pode vir depois da declaração (e também do seu uso), mas deve ser coerente:

```
double divide(double a, double b)
{
   return a / b;
}

int lg(long N) {
   int i;
   for (i = 0; N > 0; i ++, N /= 2);
   return i;
}
```

Linguagem C – Funções

- Funções podem ser recursivas:

```
long fatorial(int n)
{
   if (n <= 2)
      return (long) n;
   return n * fatorial(n – 1);
}

int lg(long N) {
   if (N <= 0)
      return 0;
   return 1 + lg(N / 2);
}
```

Linguagem C – Entrada e Saída

- Usa funções da biblioteca `<stdio.h>`
  - `printf()` – imprime texto formatado na saída padão (ponteiro de arquivo `stdout`)
  - `scanf()` – lê dados formatados da entrada padrão (ponteiro de arquivo `stdin`)
- Outras: `getc(), ungetc()`
- Para manipular dados em arquivos:
  - `fopen(), fclose(), fprintf(), fscanf(), fread(), fwrite(), fgetc().`

Linguagem C – Entrada e Saída

- Usa funções da biblioteca `<stdio.h>`
  - `printf()` – imprime texto formatado na saída padão (ponteiro de arquivo `stdout`)
  - `scanf()` – lê dados formatados da entrada padrão (ponteiro de arquivo `stdin`)
- Outras: `getc(), ungetc()`
- Para manipular dados em arquivos:
  - `fopen(), fclose(), fprintf(), fscanf(), fread(), fwrite(), fgetc().`

Linguagem C – Saída

```
printf(texto-formato, arg1, arg2, ...);
```

Cada `%` na string de texto-formato corresponde a um argumento adicional do `printf`. Exemplo:

```
int var_x = 7;
printf("x = %d\n", var_x);

int x = 10, y = 20;
printf("x = %d, y = %d\n", x, y);
```

Linguagem C – Saída

- Especificadores de formato:
  - `%d` – `int`
  - `%ld` – `long`
  - `%f` – `float`
  - `%lf` ou `%g` – `double`
  - `%c` – `char`
  - `%s` – *`string`* (`vetor de char`)
  - `%%` – `imprime o próprio %`
- O tipo de cada parâmetro extra do printf deve casar com a entrada `%` correspondente.

Linguagem C – Saída

- Exemplo:

```
char c = 101, nome[] = "Gonçalves";
double peso = 82.4;
int idade = 25;
printf("Meu nome é %s, tenho %d anos %c
peso %lf kilos.\n", nome, idade, c,
peso);
```

- O trecho de código acima imprime: `"Meu nome é Gonçalves, tenho 25 anos e peso 82.4 kilos."`
- Obs: não pode quebrar a linha do `printf`!

Linguagem C – Saída

- Há outros formatos para inteiros:
  - `%o` ou `%lo` (imprime em octal)
  - `%x` ou `%lx` (imprime em hexadecimal)
- O especificador de formato também admite parâmetros adicionais. Por exemplo, a chamada

```
printf("%04d; %.2lf%%", 97, 23.3487234);
```

imprime na saída a string "0097; 23.35%"

Linguagem C – Entrada

```
scanf(texto-formato, &var1, &var2, ...);
```

- Cada `%` do texto-formato deve corresponder a uma variável extra (do tipo especificado) a ser lida pelo scanf.
- Exemplo:

```
int x, y;
double z;
scanf("%d %d %lf", &x, &y, &z);
```

Linguagem C – Entrada

- A função scanf pula espaços em branco da entrada enquanto tenta ler os dados pedidos (ou seja, ' ', \n', '\t' e '\r' são ignorados).
- Por exemplo, se sua entrada é:

```
234      32
74.349
```

Após a chamada do scanf tem-se:

```
X = 234, y = 32 e z = 74.349.
```

Sintaxe comum de C e Java

- Operadores:
  - aritméticos:
    - binários: +, -, *, /, %
    - unários: -, ++, --
    - de atribuição: +=, -=, *=, /=, %=
  - relacionais: <, >, <=, >=, ==, !=
  - lógicos: &&, ||, !, (… ? … : …)
  - de bits: &, |, ^, <<, >>, ~

Recordação (ou não)

```
int n = 9;
```

- Se fizermos `m = (n ++);` o valor de `m` será …
- Se fizermos `m = (++ n);` o valor de `m` será …
- Se `m = (n < 7 ? 2 : 4);` o valor de `m` é …
- Se fizermos `m = n % 7;` o valor de `m` será …

13 & 3	13 | 3	13 ^ 3	13 >> 1	13 << 1
1	15	14	6	26

Sintaxe comum de C e Java

- `if ( condição ) { ... } else { ... }`
- `while ( condição ) { ... }`
- `do { ... } while ( condição )`
- `for (i = 0; i < 100; i ++) { ... }`
- `switch ( expressão ) { case 0: ... }`
- `break*, continue, return`

* (Em Java: `break line;` em C: `goto line;`)