ICMC USP

Introdução à Ciência da Computação

Aula: Conceitos Iniciais da linguagem C

Prof. Alneu de Andrade Lopes
Slides iniciais preparados pela
Profa Renata Pontin Mattos Fortes

Introdução

- Linguagens de Programação
- Fatores de Qualidade em Software
- Estilos de Programação
- Manutenção em Software
- Histórico da Linguagem C
- Estrutura de um programa em C
- Exemplo de programa C

Linguagens de Programação

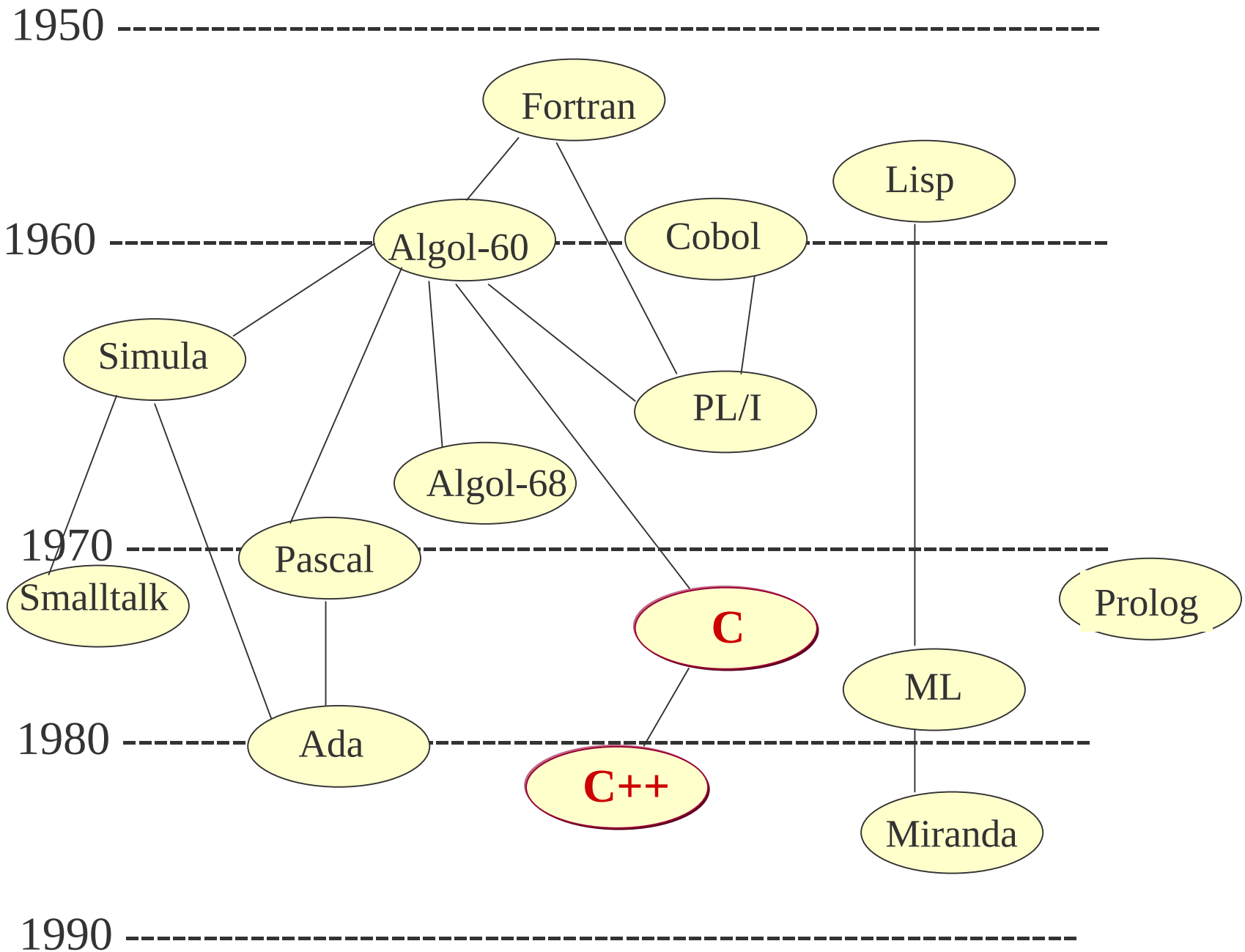
- Linguagem de Máquina (código binário)
  - inicialmente, programadores trabalhavam diretamente com 1's e 0's: 000110011 = load, ...
- Assembler (simbólico)
  - uso de mnemônicos para as instruções: LD = load
  - labels e variáveis
- Fortran (engenharia, aritmética)
  - linguagem não estruturada
  - ótimos compiladores para aritmética

Linguagens de Programação (cont.)

- Lisp (processamento simbólico - *funcional*)
  - valor de uma expressão depende apenas de suas subexpressões
  - originalmente ***sem*** atribuições
  - (lambda (*x*) (* *x x*)) => x^2
- Programação em Lógica - Prolog
  - baseada em relações
  - regras de produção:
    - *avô(João,José) :- pai(João, X), pai(X, José);*

Linguagens de Programação (cont.)

- Programação Algorítmica (imperativa)
  - módulos, rotinas, *sem goto's*
  - atribuições de variáveis e estruturas de dados
  - fluxo de controle
  - ex: Algol, Pascal, **C**
- Orientação a Objetos
  - objetos: dados + funções
  - herança, encapsulamento de dados, polimorfismo
  - ex: C++, Smalltalk, Java


1950
Fortran
Lisp
1960
Algol-60
Cobol
Simula
PL/I
Algol-68
1970
Pascal
Smalltalk
C
Prolog
ML
1980
Ada
C++
Miranda
1990

Fatores de Qualidade em Software

- O que é um bom software?
- Que fatores influenciam ou determinam a qualidade de um programa?
- Um programa que funciona é um bom programa?

Fatores externos e internos em qualidade de software

Fatores Externos (usuário)

- Facilidade de usar:
  - interface simples e clara
  - comandos não ambíguos
- Rapidez de execução
- Eficiência no uso de recursos (memória)
- Corretude:
  - habilidade do software de executar corretamente as tarefas definidas através de especificações e requimentos

Fatores Externos (usuário)

- Portabilidade:
  - facilidade de transportar o software para outras plataformas
- Robustez:
  - capacidade do software de executar em condições anormais ou inesperadas
- Integridade:
  - capacidade de autoproteção

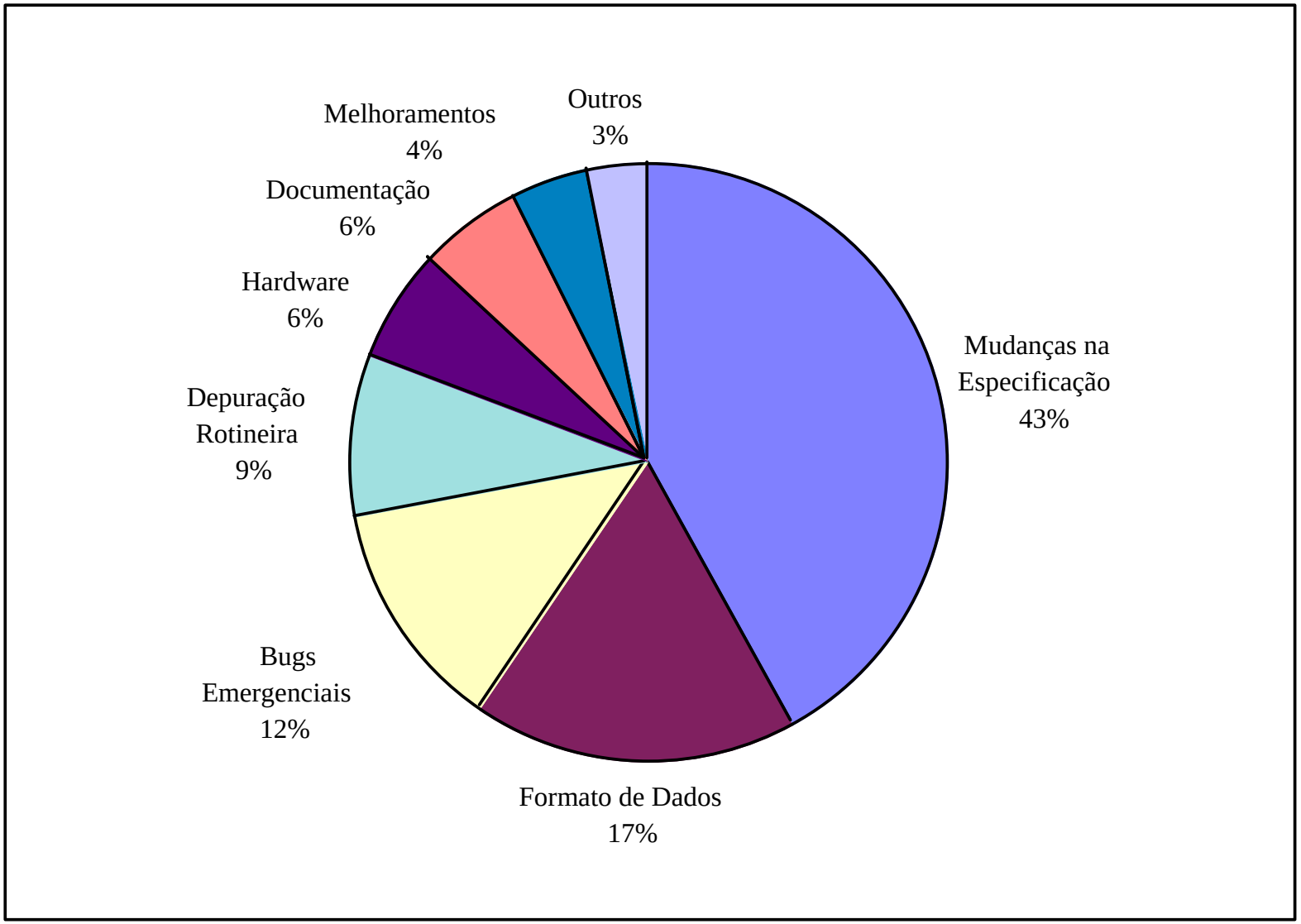
Fatores Internos (programador)

- Legibilidade:
  - facilidade de se entender o código
- Reusabilidade:
  - facilidade de se reutilizar o software desenvolvido em novas aplicações
- Modularidade:
  - sistema dividido em unidades conceituais que encapsulam informações relacionadas

Fatores Internos (programador)

- Extensibilidade:
  - facilidade de introduzir novas funções e adaptar o software a mudanças nas especificações
- Testabilidade:
  - facilidade com que o software pode ser depurado

Manutenção de Software



Estilos de Programação

- Otimização de código (assembler):
  - escovando bits para aumentar performance, reduzir tamanho
    - microcontroladores
    - drivers
- Programação estruturada:
  - centrada no algoritmo
  - abordagem descendente (top-down)
  - abordagem ascendente (botton-up)

Estilos de Programação (cont.)

- abordagem descendente (top-down) :
  - decompor a tarefa principal em subtarefas
  - refinar as subtarefas até encontrar tarefas mais simples
  - codificar
    - código dedicado ao problema
- abordagem ascendente (botton-up) :
  - implementar funções simples, de uso geral
  - compor o programa a partir destas
    - favorece reusabilidade (bibliotecas)

Estilos de Programação (cont.)

- Programação modular
  - centrada nos dados
  - módulos contém dados e procedimentos
  - encapsulamento, abstração, hierarquia

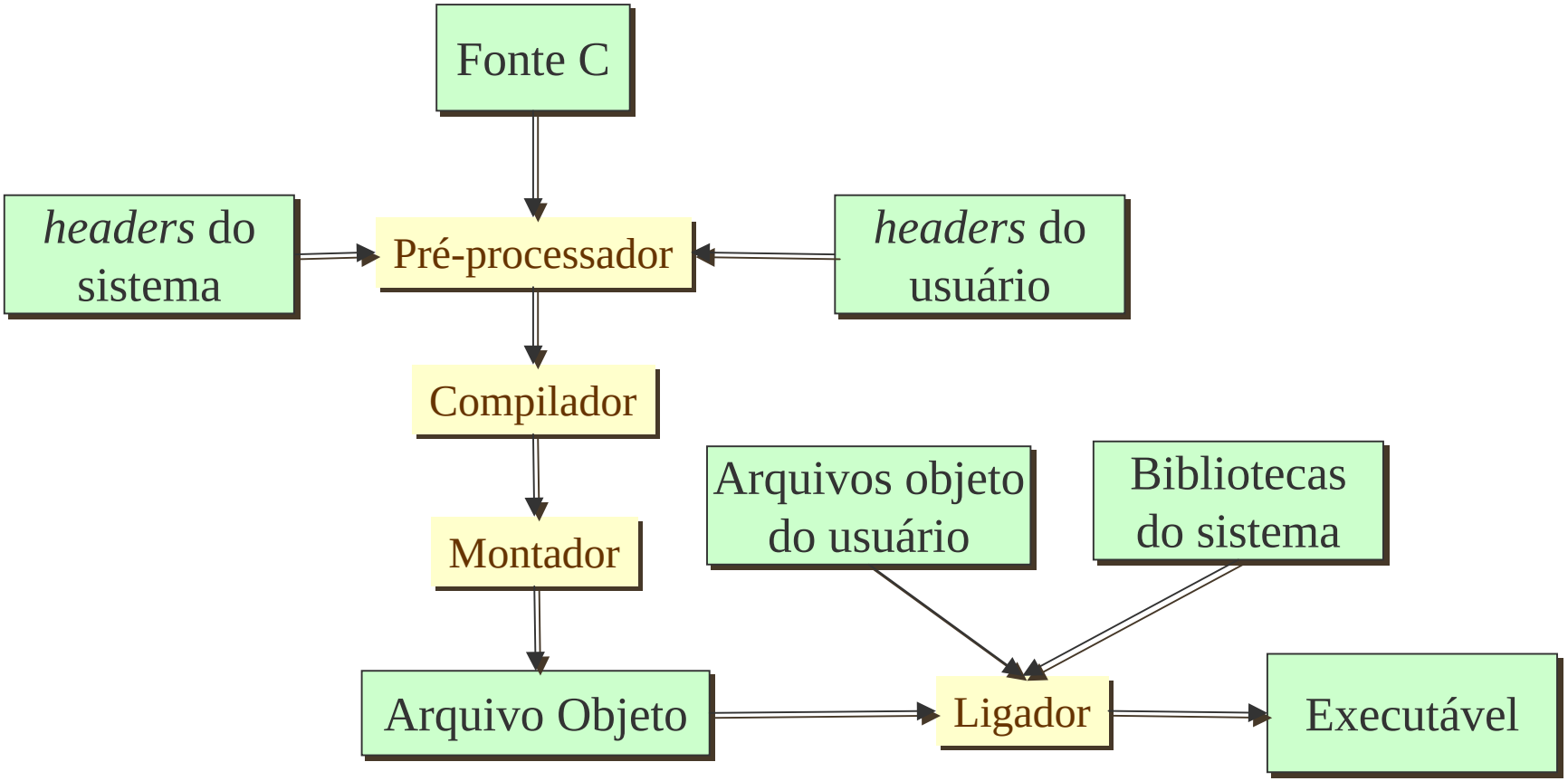
Histórico -"A" Linguagem C

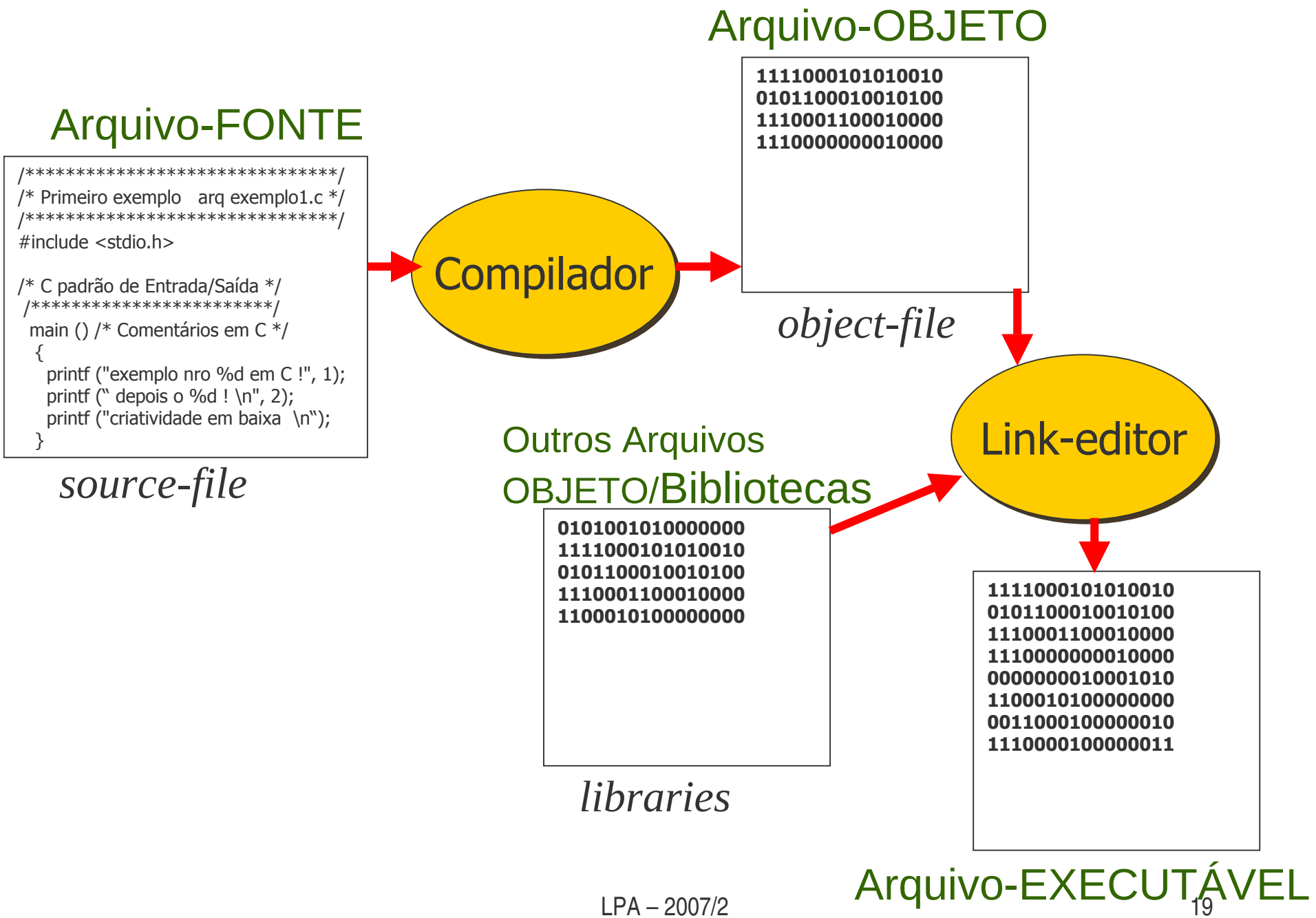
- Origem de C está associada ao sistema Unix
- BCPL: desenvolvida por Martin Richards
- 1970: Ken Thompson desenvolve B, baseada em BCPL, para o primeiro Unix no DEC PDP-7
- 1972: Dennis Ritchie projeta a linguagem C, baseado em B.
  - O sistema operacional Unix, o Compilador C e quase todos os aplicativos Unix são escritos em C!!!
- 1988: o *American National Standard Institute* (ANSI) define o padrão ANSI C

Características Gerais

- linguagem de nível *médio*
- não é uma linguagem fortemente tipada
- uso intensivo de ponteiros
- definição de blocos { }
- pré-processador
- não define operações de entrada e saída
- funções retornam valor e podem ser chamadas recursivamente

Fluxo do Compilador C




Arquivo-OBJETO
1111000101010010
0101100010010100
1110001100010000
1110000000010000
Arquivo-FONTE
/*******************************/
/* Primeiro exemplo arq exemplo1.c */
/*******************************/
#include <stdio.h>
/* C padrão de Entrada/Saída */
/*************************/
main () /* Comentários em C */
{
printf ("exemplo nro %d em C !", 1);
printf (" depois o %d ! \n", 2);
printf ("criatividade em baixa \n");
}
source-file
Compilador
object-file
Link-editor
Outros Arquivos
OBJETO/Bibliotecas
0101001010000000
1111000101010010
0101100010010100
1110001100010000
1100010100000000
libraries
1111000101010010
0101100010010100
1110001100010000
1110000000010000
0000000010001010
1100010100000000
0011000100000010
1110000100000011
Arquivo-EXECUTÁVEL

Estrutura de um Programa C

Programa C

- Diretivas ao Pré-Processador
  - Includes
  - Macros
- Declarações Globais
  - Funções
  - Variáveis
- Definição das Funções

```
main ()
{ /* begin */
} /* end */
```

Exemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define FALSE 0 /* define  F  igual 0 */
#define TRUE 1 /* define  T igual 1 */

int i = 0;
void mostra_msg( void );

main( ) {
  int resposta;
  printf( "Quer ver a mensagem?\n" );
  scanf( "%d", &resposta );
  if( resposta == TRUE )  mostra_msg( );
  else puts( "Goodbye for now." );
}
void mostra_msg( void ) {
  clrscr( );
  for( i = 0; i <=10; i++ )
  printf( "Teste  # %d.\n", i );
}
```

Exemplo (cont.)

```
#include   <stdio.h>
#include   <stdlib.h>
#include   <conio.h>
#define  FALSE 0      /* define macro F igual 0 */
#define  TRUE 1       /* define macro T igual 1 */
```

Diretivas ao Pré-processador

#**include** <filename>

- indica ao pré-processador para incluir o arquivo filename antes da compilação
- os arquivos terminados em ".h" são chamados *headers* (ou cabeçalhos). Armazenam informações globais como declaração de tipos, funções e definição de macros

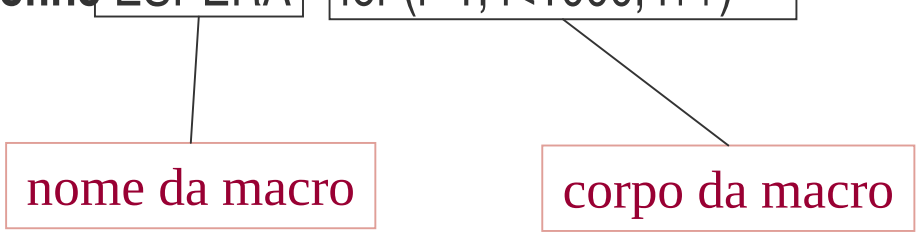
Exemplo (cont.)

#**define** FALSE 0

- define uma macro, que permite substituir um *string* por outro no corpo do programa *antes* da compilação se realizar
- no exemplo acima o pré-processador substitui as ocorrências de FALSE por 0 e TRUE por 1
- Ex:

if (resposta == TRUE) ==> **if** (resposta == 1)

#**define** ESPERA for (i=1; i<1000; i++)



Exemplo (cont.)

```
int i = 0;
void mostra_msg( void );
```

Declarações Globais

- indica ao compilador que, *dentro do arquivo* onde aparecem estas declarações:
  - a variável *i* é inteira, iniciada com o valor zero (0)
  - a função mostra_msg não recebe nenhum parâmetro e não retorna nenhum valor (procedure)
- Ex:

  int soma(int x, int y);
  - *soma* é uma função que recebe dois argumentos inteiros e retorna um valor também inteiro
  - as declarações são utilizadas pelo compilador para verificar se as funções e as variáveis globais são utilizadas corretamente

Exemplo (cont.)

- Ex: (cont.)

```
float f;
int a;
int soma (int x, int y);
...
soma (f, a);   ==> erro: parâmetro inválido
```

- mostra_msg ( ) não necessita de argumento pois utiliza uma variável global *i*.

É fortemente recomendada a não utilização de variáveis globais

Exemplo (cont.)

```
main( )
{ int resposta;
  printf( "Quer ver a mensagem? (0-no, 1-yes)\n" );
  scanf( "%d", &resposta );
   if ( resposta == TRUE )
        mostra_msg( );
   else
        puts( "Goodbye for now." );
}
```

Função Principal

- *todo* programa C tem que ter uma função chamada **main( )**. É aqui que inicia a execução do programa
- em um programa pequeno, todo o algoritmo pode ser escrito em *main()*
- programas estruturados consistem em uma hierarquia de funções dentre as quais *main()* é aquela de mais alto nível

Uma lista de palavras-chave de C ANSI

(repare, são 32 somente !!!)

auto break case char const

continue default do double else

enum extern float for goto if int

long register return short signed

sizeof static struct switch typeof

union unsigned void volatile while

Entrada e Saída Elementar

- C utiliza o conceito de *streams* (canais) tanto para realizar E/S como para acessar arquivos
- Canais pré-definidos:
  - *stdin*: associado ao teclado para entrada de dados
  - *stdout*: associado a tela para exibição de dados
  - *stderr*: mensagens de erro, enviadas a tela por *default*

Tipos de Dados

TABLE 1-1 ASCII codes

	0	1	2	3	4	5	6	7	8	9
0	\000	\001	\002	\003	\004	\005	\006	\a	\b	\t
10	\n	\v	\f	\r	\016	\017	\020	\021	\022	\023
20	\024	\025	\026	\027	\030	\031	\032	\033	\034	\035
30	\036	\037	*space*	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	\177		

Função getc( )

- int getc(FILE *stream); /* stdin.h */
  - lê um caracter do stream especificado e retorna um inteiro contendo o seu código ASCII
- Ex:

```
#include <stdio.h>
int ch;
...
ch = getc( stdin );
if ch < '8' ...
```

Função getchar( )

- int getchar(); /* stdio.h */
  - lê um caracter do teclado especificado e retorna um inteiro contendo o seu código ASCII
- Ex:

```
#include <stdio.h>
int ch;
...
ch = getchar();
if ch < '8' ...
```

Função getch( )

- int getch(); /* stdio.h */
  - lê um caracter do teclado especificado e retorna um inteiro contendo o seu código ASCII
  - não espera que o usuário tecle <*return*>
- Ex:

```
#include <stdio.h>
int ch;
...
ch = getch();
if ch < '8' ...
```

Função putc( )

- int putc(int c, FILE *stream); /* stdin.h */
  - escreve o caracter *c* no stream especificado e retorna um inteiro contendo o seu código ASCII se a operação teve sucesso, ou a constante EOF caso contrário. Ex:

```
#include <stdio.h>
int ch;
...
for ( ch = 65; ch <= 90; ch++) putc( ch, stdout);

resultado:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Função putchar( )

- int putchar(int c); /* stdio.h */
  - escreve o caracter *c* no monitor (saida padrão) e retorna um inteiro contendo o seu código ASCII se a operação teve sucesso, ou a constante EOF caso contrário
- Ex:

```
#include <stdio.h>
int ch;
...
for ( ch = 65; ch <= 90; ch++) putchar(ch);
```

```
resultado:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Entrada Formatada: scanf( )

- *scanf()* lê um string de entrada, converte os dados de acordo com a especificação de formato e armazena-os nas variáveis indicadas
- Formato:

  scanf("<formato e texto>", endereço_variáveis);

  - para se armazenar os valores lidos, são passados os endereços das variáveis, de forma que scanf saiba onde colocar os dados

Entrada Formatada: scanf( )

- Exemplo:
  - leitura de um inteiro do teclado:

```
#include <stdio.h>
void main() {
int i;
   scanf("%d", &i);
}
```

- o operador "&" localiza a variável *i* para *scanf*
- "%d" indica que o dado a ser lido é um inteiro

Saída formatada: printf( )

- *printf()* escreve um conjunto de dados na saída, convertendo os dados de acordo com a especificação de formato. Note que, ao contrário de scanf, os valores das variáveis são fornecidos

- Formato:

printf("<formato e texto>", variáveis);

Saída formatada: printf()

- Ex:

```
int i = 10;
float r = 3.1514;
char s[] = "Blablabla";  /* cadeia de caracteres */

printf("Inteiro: %d, Real: %f, String: %s",i,r,s);
```

- produz:

```
Inteiro: 10, Real: 3.151400, String: Blablabla
```

Primeiro Programa

- Programa que imprime: “Programo, logo existo”

```
#include <stdio.h>
main( )
{
 printf(“Programo, logo existo!\n”);
}
```

Exercícios

1. fazer um programa que leia um caractere do teclado e imprima-o na tela.
2. fazer um programa que lê dois inteiros do teclado e imprime a soma deles na tela
3. criar uma função que recebe dois inteiros como parâmetros e retorna sua soma
4. fazer uma função que lê um dígito entre ‘0’ e ‘9’ e imprime o caractere lido; caso o caractere lido não seja um dígito, escreva uma mensagem!

Exercício - para próxima aula

- Implementar seu primeiro programa C que:

Utilizando as funções codificadas nos exercícios anteriores, peça que sejam fornecidos 2 números inteiros (verifique se são dígitos) e apresente a soma dos mesmos!

OBS: o programa deve ser legível, documentado e testado.

Entregar impresso o código-fonte.

Fim