Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

Introdução à Linguagem C

Material preparado pela profa

Silvana Maria Affonso de Lara e utilizado por outros professores (Rosana Vaccari)

2º semestre de 2010

Roteiro da Aula

- Revisão: Algoritmos e Programas
- 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

Revisão: Algoritmos e Programas

- Conceito central de programação

  → ALGORITMO

- Programar é basicamente construir algoritmos

- Algoritmo: seqüência de comandos ou instruções que devem ser executados passo a passo em uma ordem específica

- Outro aspecto fundamental

  → ESTRUTURAS DE DADOS

Revisão: Algoritmos e Programas

- Principal dificuldade no entendimento de algoritmos
  - Como visualizar as estruturas dinâmicas das possíveis execuções do algoritmo a partir da estrutura estática do texto do algoritmo.

  - Exemplo:

```
início {começo do algoritmo}
    real: N1, N2, N3, media; {declaração das variáveis}
    {corpo do algoritmo}
    escreva('Digite as 3 notas: ');
    leia(N1, N2, N3);
    media ← (N1+N2+N3)/3;
    escreva('A média é : ', media);
fim. {término do algoritmo}
```

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

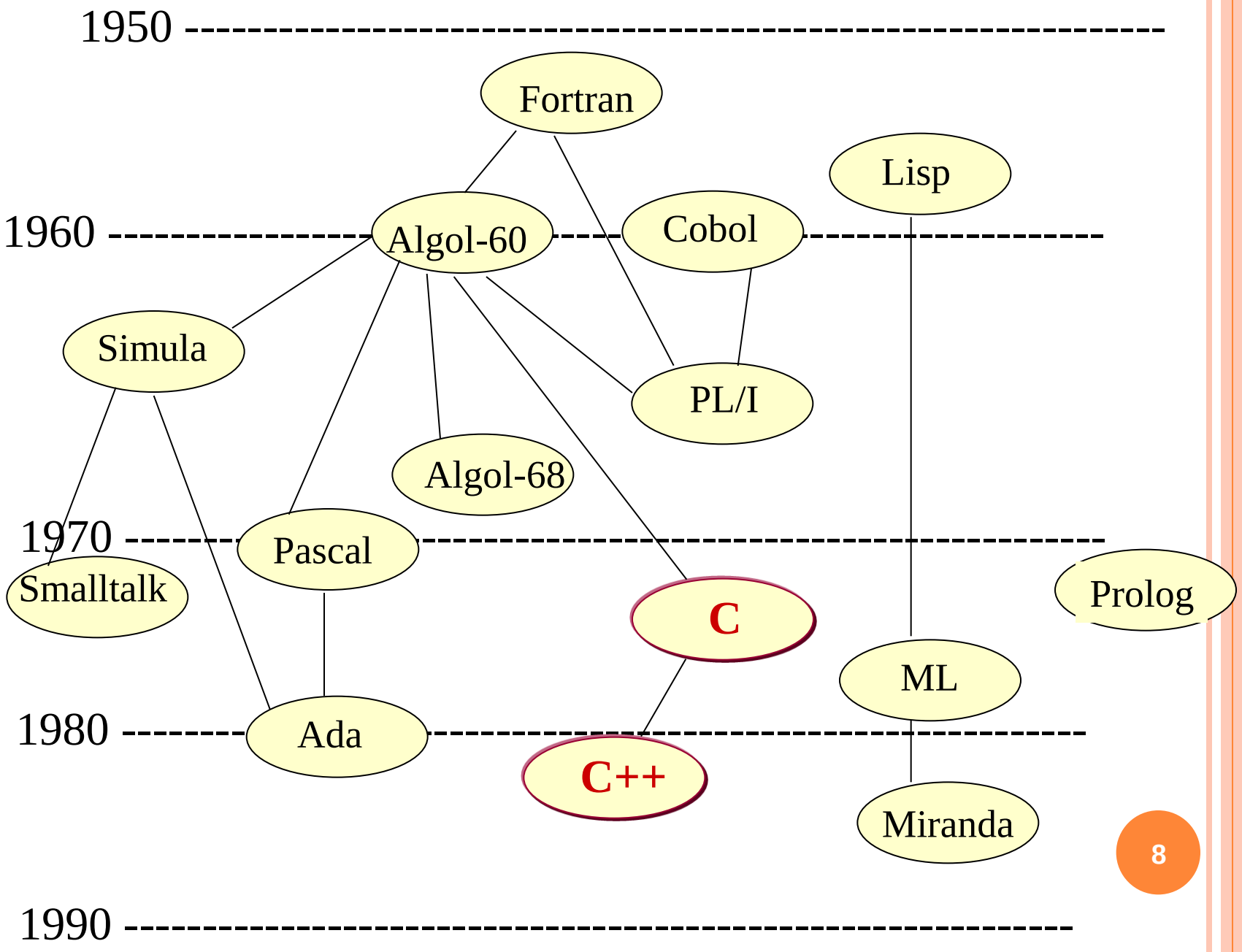
Linguagens de Programação

- 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

- 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 requerimentos

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 auto-proteção

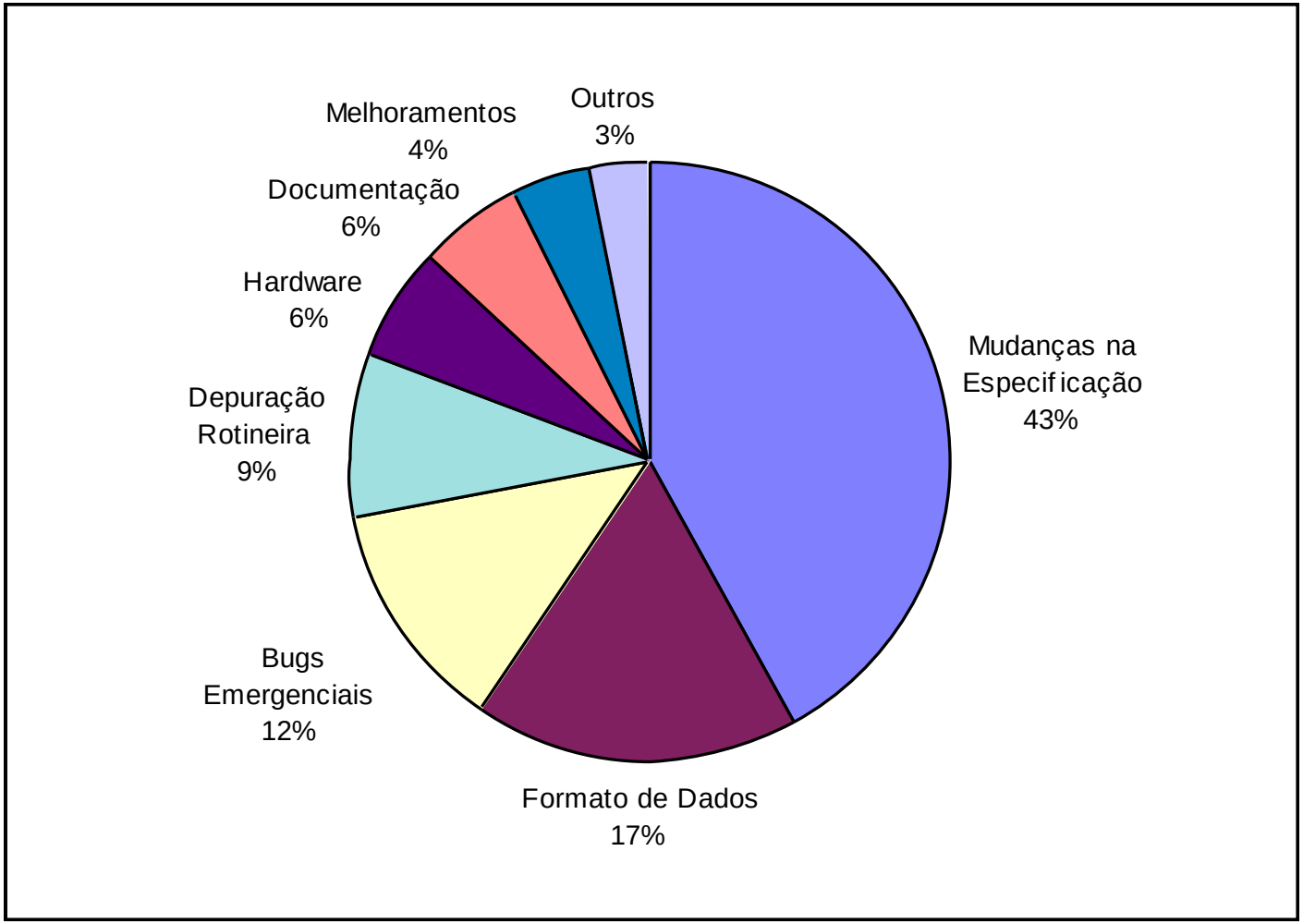
Fatores Internos (programador)

- 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 (bottom-up)

Estilos de Programação

- 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 (bottom-up) :
  - implementar funções simples, de uso geral
  - compor o programa a partir destas
    - favorece reusabilidade (bibliotecas)

Estilos de Programação

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

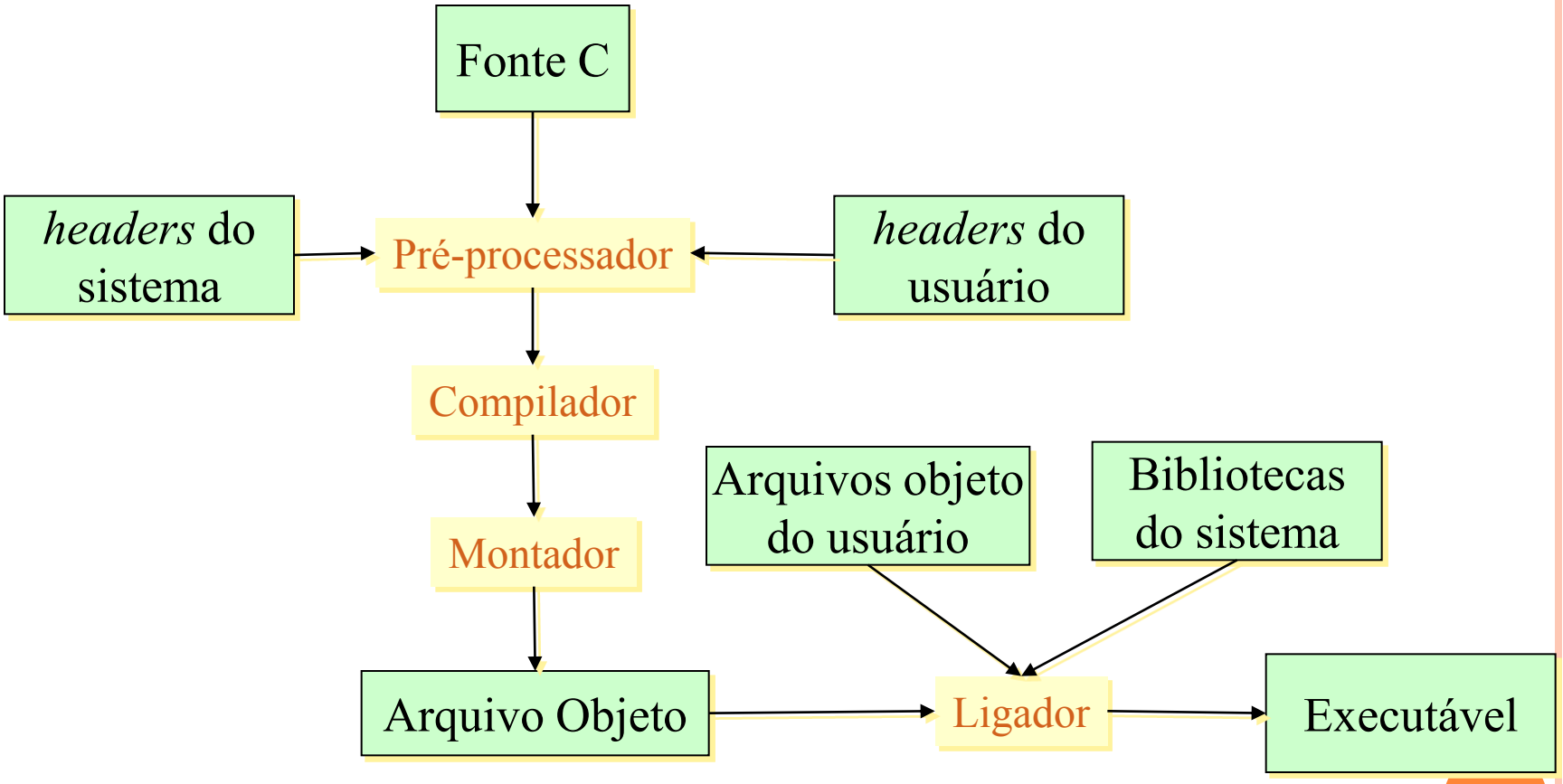
Histórico – "A Linguagem C"

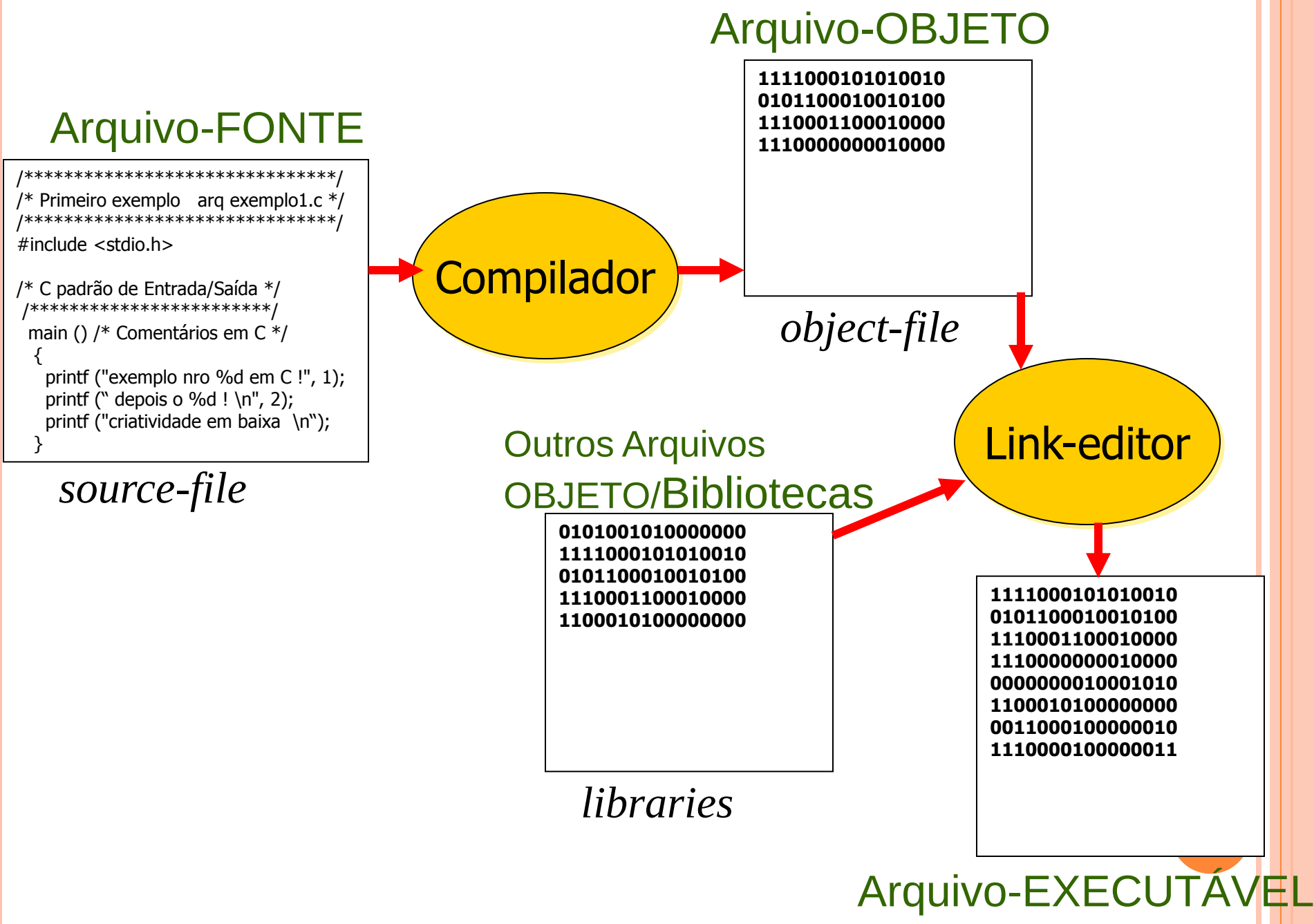
- Origem de C está associada ao sistema Unix
- BCPL: desenvolvida por Martin Richards
- 1970: Ken Thompson desenvolve B, baseada em BCPL, para o primeiro Unix no DEC PDP-7
- 1972: Dennis Ritchie projeta a linguagem C, baseada na linguagem B.
  - O sistema operacional Unix, o Compilador C e quase todos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

Características Gerais

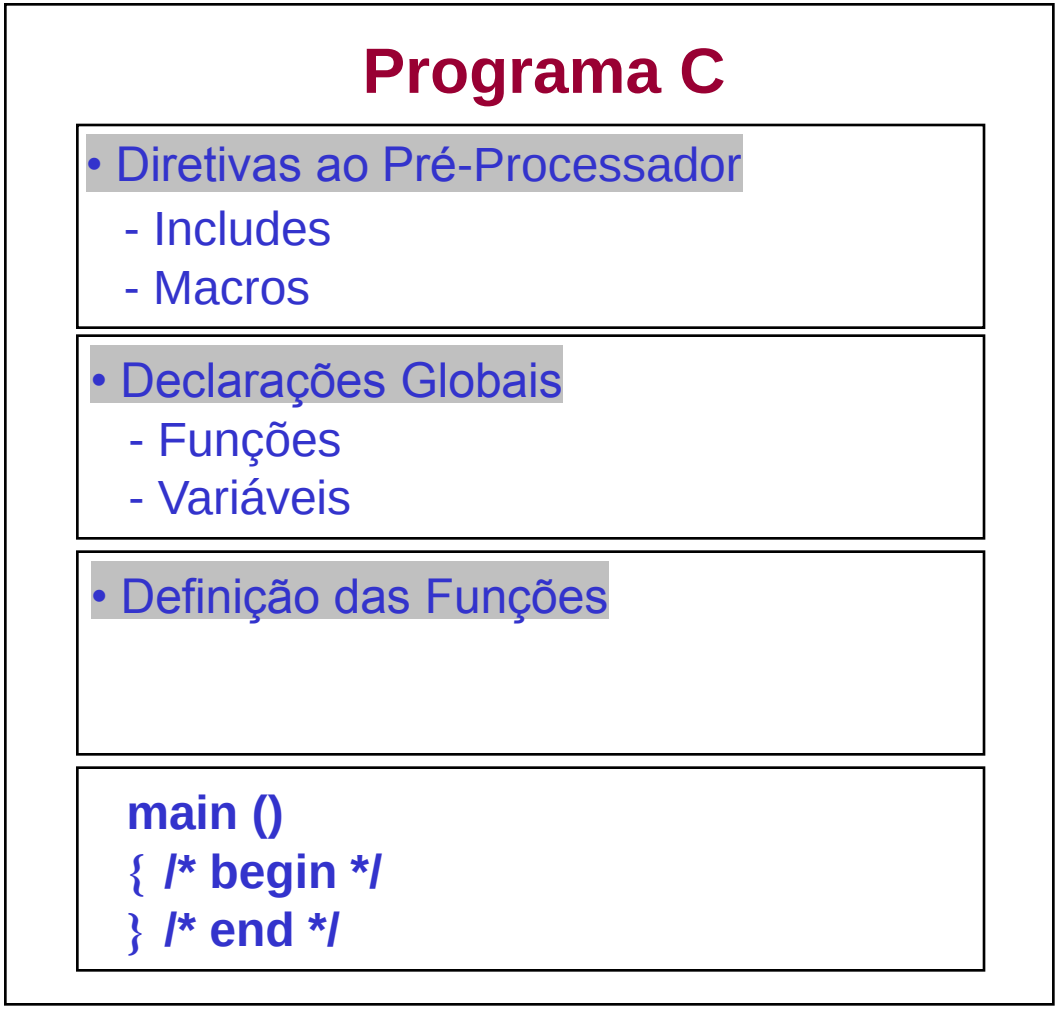
- linguagem de nível *médio*
- não é uma linguagem fortemente tipada
- uso intensivo de ponteiros
- definição de blocos { }
- pré-processador
- funções retornam valor e podem ser chamadas recursivamente

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



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 );

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 - EXPLICAÇÃO

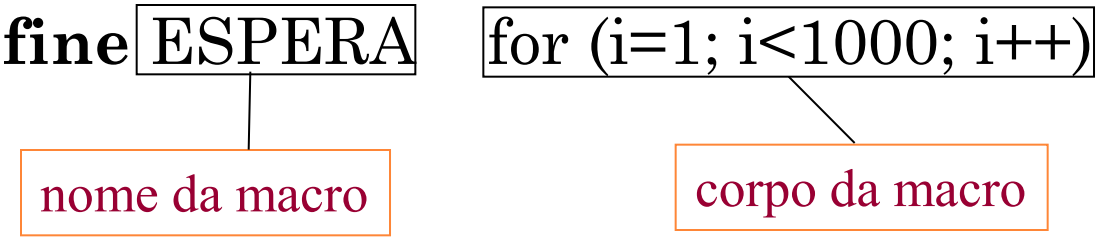
```
#include   <stdio.h>
#include   <stdlib.h>
#include   <conio.h>
#define    FALSE 0      /* define macro F igual 0 */
#define    TRUE 1       /* define macro T igual 1 */
```

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 - EXPLICAÇÃO

- **#define** FALSE 0
- define uma macro, que permite substituir um *string* por outro valor 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 - Explicação

```
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 - Explicação

- Exemplo (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

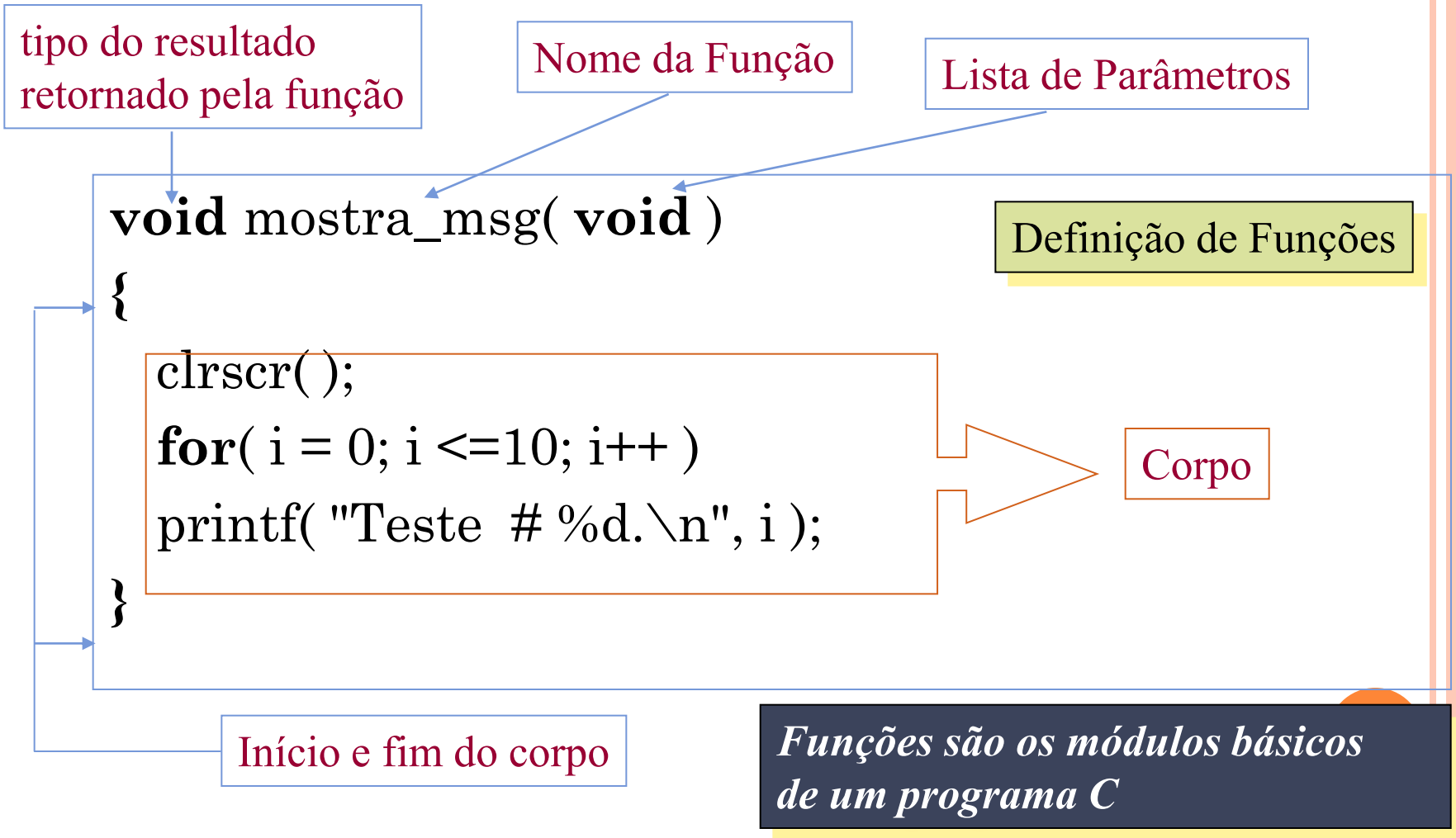
Exemplo - Explicação

```
void  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

Exemplo - Explicação



Exemplo - Explicação

- O comando ***if***

```
if ( resposta == TRUE )
  mostra_msg( );
else
  puts( "Goodbye for now." );
```

- Formato:
  - *if (<teste>)*
  - *<comando1>*
  - *else*
  - *<comando2>*

se <teste> avalia para verdadeiro, <comando1> é executado, senão <comando2> é executado

Exemplo - Explicação

- O comando ***for***:

```
for( i = 0; i <=10; i++ )
  printf( "Teste  # %d.\n", i );
```

- Formato:

for *(<atrib1> ; <teste>; <atrib2>)*
<comando>

1. executa *atribuição 1*
2. avalia *teste*: se falso, encerra *for*
3. se verdadeiro:
3.1 executa *comando*
3.2 executa *atribuição 2*
3.3 vai para 2

Exemplo - Explicação

- 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*

Exemplo - Explicação

- int puts(char *s); /* stdio.h */
  - escreve a string s no monitor (saida padrão) e retorna um inteiro não negativo se a operação teve sucesso, ou a constante EOF caso contrário

- Ex:

```
#include <stdio.h>
...
puts("Isto é uma string!");
```

```
resultado:
Isto é uma string!
```

Exemplo - Explicação

- 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

Exemplo - Explicação

- 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

Exemplo - Explicação

- 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);

Exemplo - Explicação

- 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

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

EXERCÍCIOS

1. Escreva um programa em C que calcula o preço total de um produto, tendo como entrada o preço unitário e a quantidade vendida.

```
Algoritmo calculo
Inicio.
Var
    real: preco, qtde, total;
    escreva("entre com o preco do produto:");
    leia(preco);
    escreva("entre com a quantidade vendida:");
    leia(qtde);
    total = preco * qtde;
    escreva("o preco total do produto e:", total);
Fim.
```

2. Escreva um algoritmo para calcular o consumo médio de um automóvel (medido em km/l), dada a distância total percorrida e o volume de combustível consumido para percorre-la (em litros).