

Programação estruturada em

C

# Resolução de problemas com um computador

- Entender o *problema*
- Encontrar um *algoritmo* para resolvê-lo
- *Implementar* o algoritmo numa linguagem de programação

# Linguagens de programação

- Permitem implementar um algoritmo
  - Expressar o algoritmo numa forma que o computador entenda
    - Precisão, zero ambiguidades
- Implementação (ou *código*) é texto
  - Escrito segundo as regras de *sintaxe* e *semântica* de uma linguagem
  - Em AEDS 1 usaremos a linguagem C

# Linguagem C: características

- Eficiência
  - Construções similares a instruções de máquina
- Acesso direto à memória
- Portabilidade
  - De microcontroladores a supercomputadores
- Poucos requisitos para execução

# Software escrito em C

- Software de sistema ou de base
  - Linux
  - Gnome
  - Python, Perl, PHP, GCC
  - Bibliotecas
    - GNU Scientific Library
    - Partes do Matlab
- Várias aplicações

# Primeiro programa

```
#include <stdio.h>
```

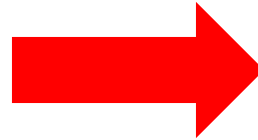
```
int main(void) {  
    printf("meu primeiro programa\n");  
    return 0;  
}
```

# Compilação

primeiro.c

```
#include <stdio.h>
```

```
int main(void) {  
    printf("primeiro\n");  
    return 0;  
}
```

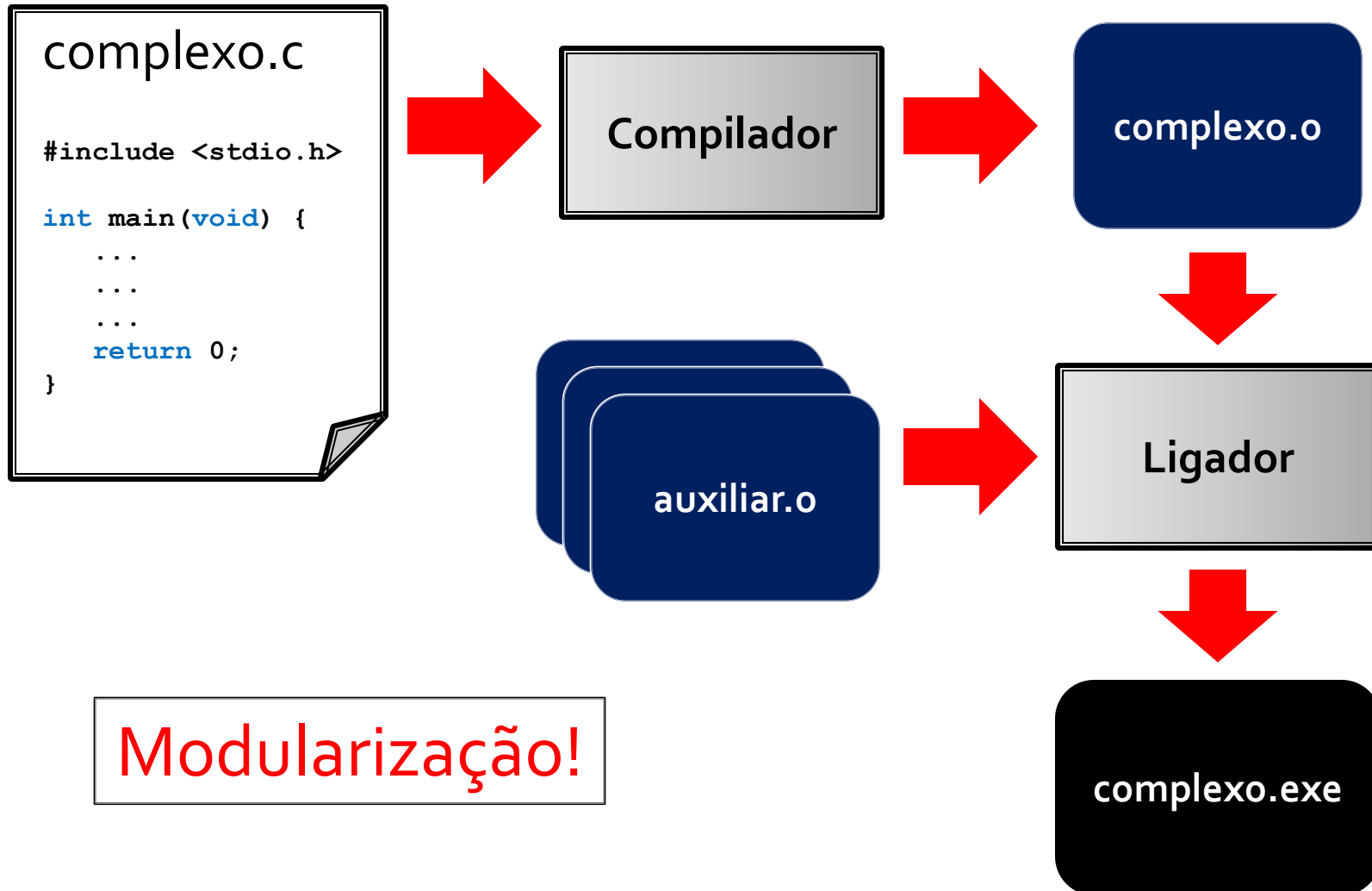


Compilador  
(GCC)



primeiro.exe

# Compilação e ligação





# Comandos e blocos

```
#include <stdio.h>
```

```
int main(void) {  
    printf("meu primeiro programa\n");  
    return 0;  
}
```

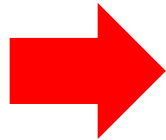
- Comando: instrui o computador a realizar uma operação
  - Todo comando termina com ponto-e-vírgula
- Bloco: agrupamento de comandos
  - Bloco delimitado por abre-e-fecha-chaves

# Primeiro programa, passo a passo

```
#include <stdio.h>
```



```
int main(void) {
```



```
printf("meu primeiro programa\n");
```



```
return 0;
```

```
}
```

# Variáveis

- No computador simplificado, armazenamos e operamos com valores nos escaninhos
- Em C, armazenamos e operamos com valores em variáveis
- Valor: armazenado na memória
- Nome: permite “usar” a variável no código

# Variáveis

- Declaração: define o *tipo* e o *nome* da variável

```
int dia;
```

```
int mes;
```

```
tipo nome;
```

- Atribuição: muda o valor de uma variável

```
dia = 14;
```

```
mes = 3;
```

- Acesso: recupera o valor de uma variável

```
dia = mes;
```

# Variáveis

- Declaração: define o *tipo* e o *nome* da variável

```
int dia;
```

```
int mes;
```

```
tipo nome;
```

- Atribuição: muda o valor de uma variável

```
dia = 14;
```

```
mes = 3;
```

- Acesso: recupera o valor de uma variável

```
dia = mes;
```

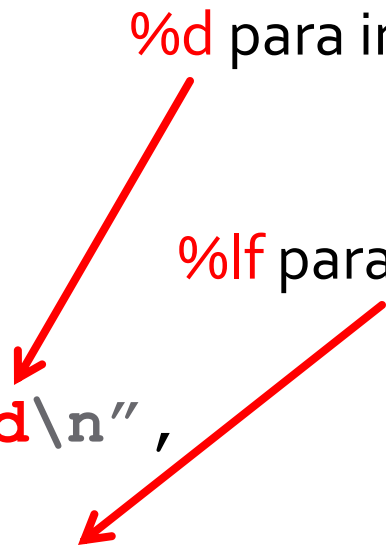
# Tipos de variáveis

- Tipos inteiros
  - `char`, `short`, `int`, `long`, `long long`
- Tipos de ponto flutuante
  - `float`, `double`, `long double`
- `void`
- Estruturas
- Arranjos
- Ponteiros

# Imprimindo variáveis

```
printf("formato", v1, v2, v3, ...);
```

```
#include <stdio.h>
int main(void) {
    int dia = 14;
    int mes = 3;
    int ano = 2013;
    double temp = 28.5;
    printf("hoje é %d/%d/%d\n",
        dia, mes, ano);
    printf("está fazendo %lf graus\n", temp);
    return 0;
}
```



`%d` para imprimir `int`

`%lf` para imprimir `double`

# Posição de uma variável

- Variáveis são armazenadas na memória
- Onde?
  - Só usar o operador & (e comercial)

```
#include <stdio.h>
```

```
int main(void) {
```

```
    double temp = 28.5;
```

```
    int dia = 14;
```


```
    printf("temp está em %p\n", &temp);
```

```
    printf("dia está em %p\n", &dia);
```

```
    return 0;
```

```
}
```

%p para imprimir  
posição de memória





# Exemplos

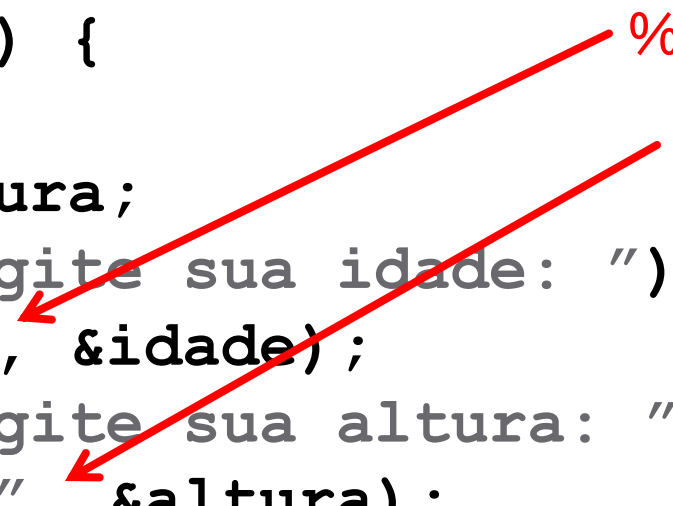
---

- `declara_imprime`
- `imprime_posicao`
- `erros_atribuicao`

# Lendo uma variável do teclado

```
scanf("formato", v1, v2, v3, ...);
```

```
#include <stdio.h>
int main(void) {
    int idade;
    double altura;
    printf("digite sua idade: ");
    scanf("%d", &idade);
    printf("digite sua altura: ");
    scanf("%lf", &altura);
    printf("%d anos e %lfm\n", idade, altura);
    return 0;
}
```



`%d` para ler `int`

`%lf` para ler `double`

# Exemplos

---

- `le_imprime`

# Detecção de erros na compilação

- Economize tempo de depuração tratando *todos* os avisos do compilador
  - Muitas vezes a mensagem de erro não reflete o que está ocorrendo; observar as redondezas da linha em que o erro/warning foi indicado

```
[debian:~/prof/aeds2/src]% gcc -Wall data.c
data.c: In function 'main':
data.c:12: warning: 'return' with no value,
           in function returning non-void
data.c:11: warning: 'hoje' is used
           uninitialized in this function
```

# Representação de inteiros sem sinal

- Inteiros são representados em binário
- O  $i$ -ésimo bit vale  $2^i$
- $13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0 = 0000\ 1101 = 0x0d$
- $44 = 32 + 8 + 4 = 2^5 + 2^3 + 2^2 = 0010\ 1100 = 0x2c$
- $64 = 64 = 2^5 = 0100\ 0000 = 0x40$
- Maior inteiro em 8 bits?
- $0xFF = 1111\ 1111 = 128 + 64 + 32 + \dots + 1 = 255$

# Representação de inteiros negativos: complemento de 2

- Representação eficiente que facilita operações aritméticas
- Em complemento de dois, inteiros negativos têm o primeiro bit igual a 1
- Maior inteiro em 8 bits:
- $0x7F = 0111\ 1111 = 64 + 32 + 16 + \dots + 1 = 127$

# Representação de inteiros negativos: complemento de 2

- Para calcular a representação de um número negativo em complemento de dois
  - Inverta os bits da representação positiva
  - Some 1
- $43 = 0010\ 1011 = 32 + 8 + 2 + 1$
- $-43 = 1101\ 0100 + 1 = 11010101$
- $1 = 0000\ 0001$
- $-1 = 1111\ 1110 + 1 = 1111\ 1111$
- $12 = 0000\ 1100$
- $-12 = 1111\ 0011 + 1 = 1111\ 0100$

# Exemplos

---

- `tamanhos_tipos`
- `limites_tipos`



# Representação de números ponto flutuante: padrão IEEE-754

- Representação com ponto fixo: 130,25
- Representação com ponto flutuante:  $1,3025 \times 10^2$
- Representação tem três partes:

- Sinal (1 bit)

- Mantissa

- Expoente

- Exemplo:

- $13.25 = 2^3 + 2^2 + 2^0 + 2^{-2} = 1101.01 = 1.10101 \times 2^3$

mantissa

expoente

# Representação de números ponto flutuante: padrão IEEE-754

- Exemplo:
  - $0.25 = 2^{-2} = 0.01 = 1.0 \times 2^{-2}$
- **float** tem 32 bits
  - 1 de sinal
  - 23 de mantissa
  - 8 de expoente
- **double** tem 64 bits
  - 1 de sinal
  - 52 bits de mantissa
  - 11 bits de expoente

# Representação de números ponto flutuante: padrão IEEE-754

- Precisão é limitada: alguns números inteiros não podem ser representados
  - float tem precisão de seis casas decimais
  - double tem precisão de dezoito casas decimais
- Representação muito diferente de números inteiros; não confundir

# Exemplos

---

- `precisao_float`



# Exemplo

---

- tabela\_ascii

# Escopo de variáveis

- Onde podemos acessar uma variável?
- O *escopo* de uma variável é o bloco onde ela foi declarada e todos os blocos dentro deste

# Exemplo

---

- escopo



# Identificadores e palavras reservadas

- Identificadores de variáveis
  - Letras, números, e *underscores*
  - Não podem começar com número

■ Palavras reservadas	auto	double	int	struct
	break	else	long	switch
	case	enum	register	typedef
	char	extern	return	union
	const	float	short	unsigned
	continue	for	signed	void
	default	goto	sizeof	volatile
	do	if	static	while
	inline	restrict		

# Expressões

- Constante

- 3.14159

- x

- Chamada de função

- `printf("chamada ao printf() \n")`

- possível efeito colateral

- Atribuição

- `x = y + z`

- `local = valor`

# Operadores

- Aritméticos
  - $x + y, x - y, x * y, x / y, x \% y, -x$
- Incremento e decremento
  - $x++, ++x, y--, --y$
- Comparação
  - $x > y, x >= y, x < y, x <= y, x == y, x != y$
- Lógicos
  - $!x, x \&\& y, x || y$
- Binários
  - $x \& y,$
- Atribuição
  - $x = y,$
- Conversão
  - $(int)x, (double)x$

Precedência

1 << 2 \* 3 % 4 ^ 5 - 6 && 7

# Lendo variáveis com scanf()

```
scanf("formato", &v1, &v2, &v3, ...);
```

```
#include <stdio.h>
int main(void) {
    double raio;
    double PI = 3.14159;
    printf("digite o raio do circulo: ");
    scanf("%lf\n", &raio);
    printf("area = %lf\n", PI*raio*raio);
    return 0;
}
```

%lf para ler double

%d para ler int

%c para ler char

passamos o *local* onde a variável está armazenada para o scanf() preencher o valor