

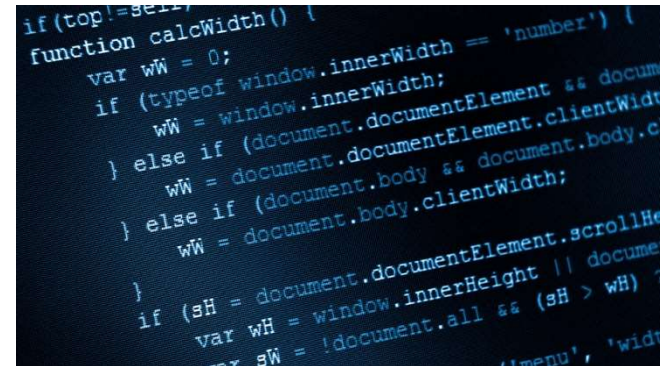
INTRODUÇÃO - LINGUAGEM C

Simone Aires - Saulo Queiroz, UTFPR-PG

Introdução

2

- ❑ Para que um algoritmo possa ser executado por um computador, ele deve ser primeiramente descrito em alguma linguagem de programação.
- ❑ Uma linguagem de programação, assim como outra linguagem qualquer, é apenas um meio de comunicação entre dois interlocutores, programadores e o processador do computador.



```
if (top !== self) {  
  function calcWidth() {  
    var wW = 0;  
    if (typeof window.innerWidth == 'number') {  
      wW = window.innerWidth;  
    } else if (document.documentElement.clientWidth) {  
      wW = document.documentElement.clientWidth;  
    } else if (document.body.clientWidth) {  
      wW = document.body.clientWidth;  
    }  
    if (sH = document.documentElement.scrollHeight) {  
      var wH = window.innerHeight || document  
      var sW = !document.all || (sH > wH) ?  
        (menu, 'wid
```

Linguagem de Programação

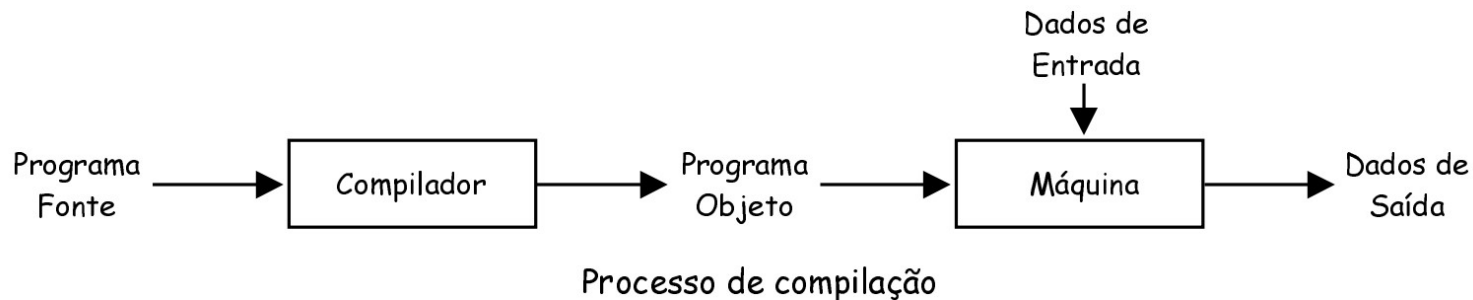
3

- Como toda linguagem, a linguagem de programação possui regras sintáticas e semânticas que devem ser seguidas para que a mensagem que se deseja passar seja compreendida
 - ▣ Regras Sintáticas → forma da escrita
 - ▣ Regras Semânticas → sentido do que se escreve

Linguagem de Programação

4

- Observe, entretanto, que mesmo utilizando uma linguagem de programação, o programa será ainda assim um texto, como uma carta e o processador do computador só entende **linguagem de máquina**, sequências de zeros e uns.



Linguagem de Programação

5

- ❑ Para fazer a tradução do programa para a linguagem de máquina do processador, existem os **Compiladores**.
- ❑ Os compiladores criam uma “versão” em linguagem de máquina do programa.
- ❑ São eles que se encarregam de “completar” e “adequar” o programa para que possa ser executado no sistema computacional desejado.

Instruções Primitivas



- São os comandos básicos que efetuam tarefas essenciais para a operação dos computadores, como entrada e saída de dados e movimentação dos mesmos na memória.
- Estes tipos de instruções estão presentes na absoluta maioria das linguagens de programação.
- Logo, com um algoritmo em mão, basta então, passá-lo para uma determinada linguagem de programação fazendo uso de sua sintaxe.

Palavras Reservadas

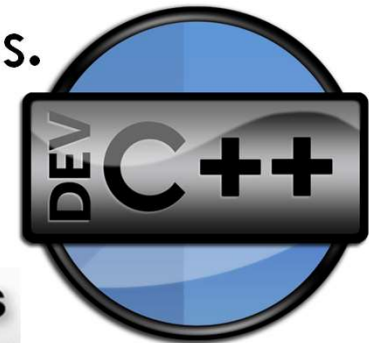
7

- São palavras que fazem parte da sintaxe de uma linguagem de programação, desta mesma forma, o comando **de entrada, saída** e o **tipo de dado** de uma variável são palavras reservadas de algoritmos e **não** podem ser usados como nome de variáveis.
- Por convenção de uso e boa prática, as palavras reservadas são destacadas nas IDE.

IDE

8

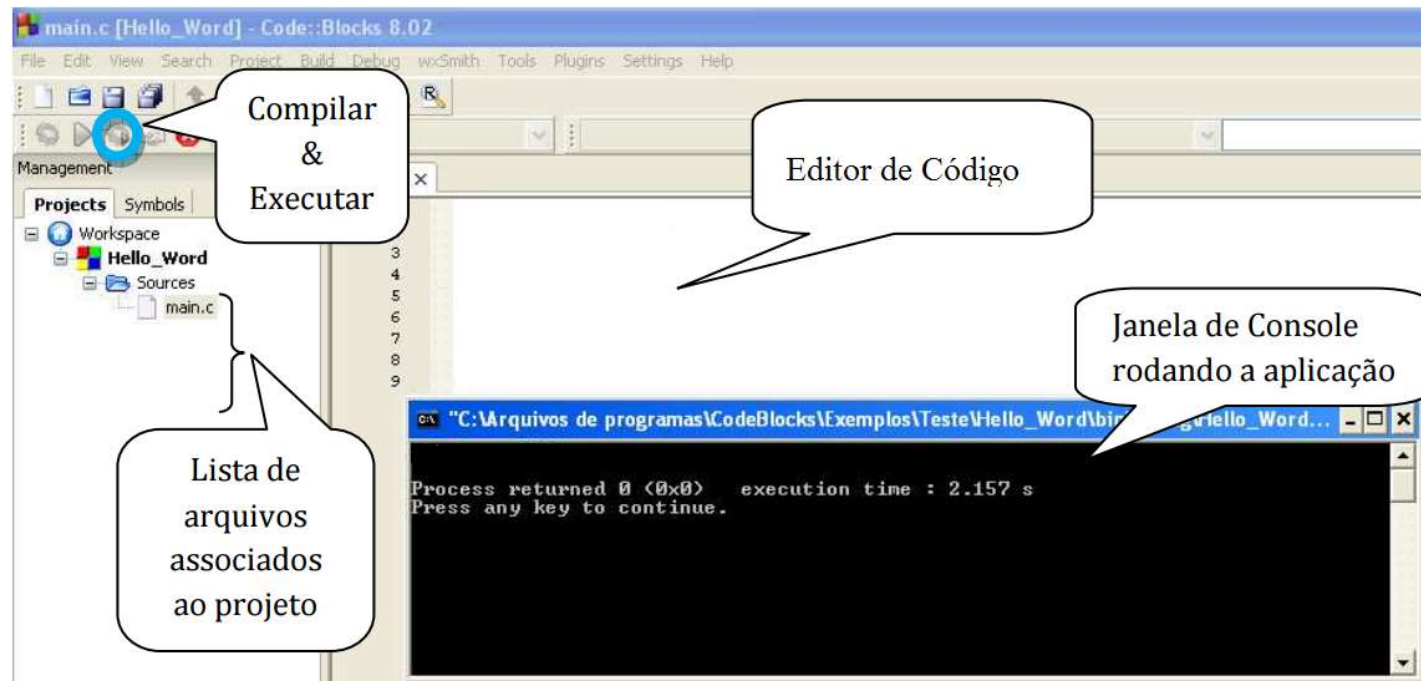
- ❑ **Ambiente de Desenvolvimento Integrado**
- ❑ Maior produtividade dos desenvolvedores.
- ❑ Normalmente consiste de:
 - ❑ Editor de código fonte;
 - ❑ Compilador;
 - ❑ Depurador.



IDE

9

Exemplo



Tipos de Linguagem de Programação

10

- Maneiras de classificação:
 - ▣ Grau de abstração;
 - ▣ Paradigma;
 - ▣ Estrutura de tipos;
 - ▣ Propósito;
 - ▣ Meios de implementação (I/C).

Grau de Abstração

11

- **Linguagem de alto nível:** é uma linguagem que se aproxima mais da linguagem utilizada pelo ser humano. Ex: Basic, Pascal.
- **Linguagem de médio nível:** combina os elementos de linguagem de alto nível com o de baixo nível. Ex: C.
- **Linguagem de baixo nível:** trata-se de linguagens de programação mais próximas ao código da máquina. Ex: Assembly.

Paradigma

12

- ❑ Paradigma estruturado;
- ❑ Paradigma orientado a objetos;

Linguagem Estruturada

13

- ❑ Os programas desse paradigma podem ser reduzidos a apenas três estruturas: **sequência, decisão e iteração**.
- ❑ Compartilhamento de códigos e dados.
- ❑ Emprego de sub-rotinas.
- ❑ Suporta diversas construções de laços.

Linguagem Estruturada

14

□ Exemplos de linguagens estruturadas e não-estruturadas:

▣ Não Estruturadas:

- Fortran
- Cobol

▣ Estruturadas:

- Pascal
- C

Linguagem de Programação C

15

- Dennis Ritchie – 1972
- Sistema Operacional UNIX
- Aplicações
- Características
 - ▣ Nível Médio
 - ▣ Linguagem estruturada
 - ▣ Portabilidade
 - ▣ Programas compilados
 - ▣ Código eficiente
 - ▣ Uso de bibliotecas

Bibliotecas

16

- Bibliotecas possuem uma série de funções prontas com recursos adicionais e disponíveis para utilização.
- A ideia por trás de biblioteca é fornecer um conjunto básico de operações, de tal forma que a portabilidade do C ANSI entre diversas plataformas seja relativamente simples.
- Além da biblioteca padrão, outras bibliotecas foram desenvolvidas para incorporar outras funcionalidades específicas na Linguagem.

Bibliotecas Comuns

17

Cabeçalhos	Descrição
ctype.h	Utilizado para testar e converter caracteres
math.h	Define várias funções matemáticas
stdio.h	Permite realizar operações de entrada/saída
stdlib.h	Define várias funções de propósito geral como gerenciamento de memória dinâmica, geração de número aleatório, comunicação com o ambiente, aritmética de inteiros, busca, ordenação e conversão
string.h	Define funções para manipulação de strings
time.h	Define funções para ler e converter datas e horas

18

CÓDIGO-FONTE

Profa. Simone Aires - Prof. Saulo Queiroz

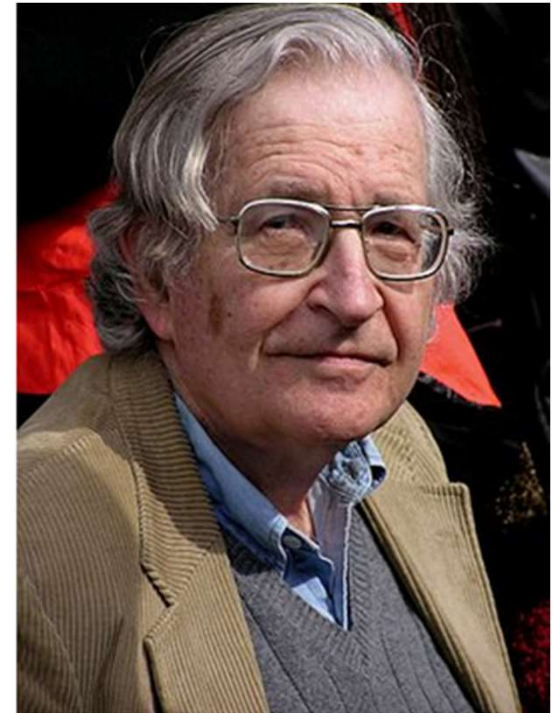
*Precisamos ressaltar as diferenças entre
linguagem natural (linguagem do dia-a-
dia) e as linguagens dos computadores!*

Linguagem Natural (LN) e Sintaxe

20

□ Noam Chomsky

- ▣ Tentou propor um formalismo matemático para a **sintaxe** das LNs
 - Sintaxe: **Regras gramaticais** que toda fala deve obedecer para uma comunicação com sucesso.
- ▣ O fenômeno da LN, porém, não é de todo domável
 - Tolerar erros na comunicação
 - Dá margem a **ambiguidades**



Profa. Simone Aires - Prof. Saulo Queiroz

LN: Ambiguidade e erros

21



- Vossa mercê, vosmecê, você, ocê, cê
- “Chame-a para mim.”, “chama ela pra eu.”
- Formalizar LNs parece impossível, porém o trabalho de Chomsky serviu para lançar os fundamentos das Linguagens de Programação (LPs)

Linguagem: Humana vs. de Programação

22

- ❑ Um código-fonte deve obedecer rigorosamente a sintaxe da LP onde ele é escrito
- ❑ Um **erro de sintaxe** impedirá o compilador de gerar o código em linguagem de máquina (programa) a partir do código-fonte
 - ▣ **Erro de compilação**

23

SINTAXE BÁSICA DE UM CÓDIGO-FONTE

Profa. Simone Aires - Prof. Saulo Queiroz

Algoritmo: Sintaxe da Estrutura básica

24

Comentário

Nome do algoritmo/sub rotina

Sequência de instruções

25

IDE - CodeBlocks

Profa. Simone Aires - Prof. Saulo Queiroz

IDE Programação

26

- ❑ Utilizarei nas aulas CodeBlocks
- ❑ Vocês **podem** utilizar a IDE que preferirem!

27

COMENTÁRIOS

Profa. Simone Aires - Prof. Saulo Queiroz

Comentários

28

- ❑ Uma instrução especial oferecida por qualquer LP
- ❑ Cria uma “área especial” onde será possível digitar qualquer texto. O compilador ignorará esse texto
 - ▣ Pode ser usada em qualquer parte do código
- ❑ Serve para vários propósitos. Por exemplo:
 - ▣ Explanarmos o código ou trechos dele, desativar uma instrução do algoritmo, lembrar a existência de *bugs*.

Comentários

29

- ❑ Ausência de comentários não geram erros de compilação. Contudo:
 - ❑ *Comentar constitui uma boa prática de programação.*
 - Ex.: códigos “fantásticos” não são aceitos em projetos open-source por falta de comentários
 - ❑ Convém pelo menos explicar o que nosso algoritmo faz

Comentários: Sintaxe

30

- Existe duas formas de comentar em Linguagem C
 - ▣ Somente uma linha
 - ▣ Um bloco que pode englobar quantas linhas quisermos

Comentários em linha

31

- Sintaxe `//<comentário>`
- Exemplos
 - ▣ `// Código feito por Fulano de tal`
 - ▣ `// Abaixo usei a fórmula de Bhaskara`
 - ▣ `// Tem um bug na linha abaixo. Corrigir`

Comentários em bloco

32

- Sintaxe `/* <comentário> */`
 - ▣ a partir de `/*` tudo será ignorado pelo compilador até `*/`
- Exemplo

```
/*
    Algoritmo por Fulano de Tal.
    Calcula as raízes de uma equação do segundo
    grau e apresenta-as na tela.
*/
```


Comentários em bloco

33

- Sintaxe `/* <comentário> */`
 - ▣ a partir de `/*` tudo será ignorado pelo compilador até `*/`
- Exemplo

```
/*
    Algoritmo by Fulano de Tal.
    Calcula a média final de um aluno. Assume que
    as notas dadas são positivas não maiores que
    10.
*/
```

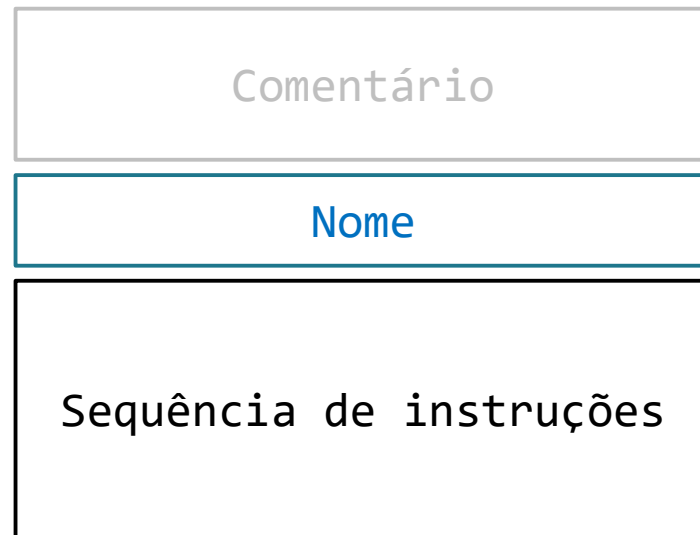
34

Algoritmo : Sintaxe do Esqueleto básico

Profa. Simone Aires - Prof. Saulo Queiroz

Algoritmo: Sintaxe do Esqueleto básico

35

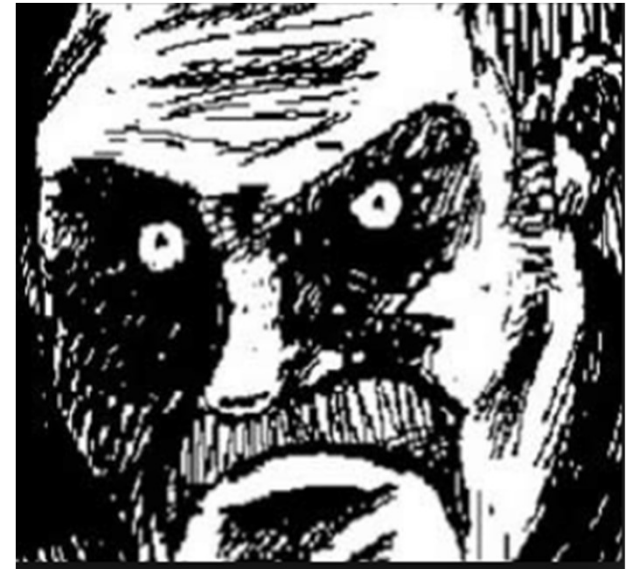


```
/* Comentário inicial */  
void SOMAR()  
{  
    /*  
    sequência de instruções  
    separadas por ponto-e-vírgula  
    */  
}
```

“Posso dar o nome que quiser ao meu algoritmo/sub-rotina?”

“Posso dar o nome que quiser ao meu algoritmo?”

NÃO



Palavras reservadas

38

- ❑ Por padrão, uma mesma palavra não pode ser usado para mais de um propósito distinto em uma **linguagem formal de programação**
- ❑ As instruções da LP já têm significado pré-definido.
 - ▣ Por isso chamam-se **palavras reservadas**.

Palavras reservadas

39

- Até agora só conhecemos:
 - ▣ Palavras: `void`
 - ▣ Símbolos: `{ , } , // , /* , */`

Palavras reservadas: Convenções

40

- *Case-sensitive*: Faz diferença entre maiúsculas e minúsculas
 - ▣ C e o bash são *case-sensitive*
 - ▣ `void` em vez de `Void`

Nomeações em algoritmos: Convenções

41

- ❑ Não podemos usar palavras reservadas
- ❑ Não podem conter caracteres especiais (exceto o caracter *underscore* _)
- ❑ Não podem iniciar com números
- ❑ Nome deve ter a ver com o que o código faz
 - ▣ **Mnemônico:** aquilo que faz lembrar ...

Nomeações em algoritmos: Exemplos

42

- Exemplos válidos:

- ▣ Soma
- ▣ MultiplicaMatriz1
- ▣ _soma
- ▣ Soma_2

- Exemplos inválidos:

- ▣ 1codigo
- ▣ *
- ▣ +algoritmo
- ▣ soma&multiplica

Instruções básicas: Entrada/Saída

Profa. Simone Aires - Prof. Saulo Queiroz

Algoritmos: Entrada e Saída

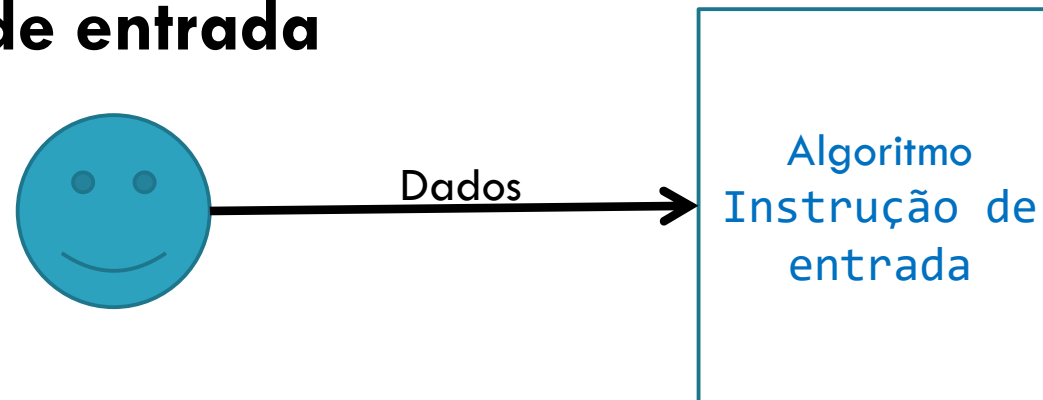
44

- Um algoritmo não apresenta muita utilidade se não puder “trocar” dados com o “mundo exterior”
- Essas trocas ocorrem de duas formas:
 - ▣ Do “mundo exterior” para o algoritmo: **Entrada**
 - Ex.: As notas de um aluno
 - ▣ Do algoritmo para o “mundo exterior”: **Saída**
 - Ex.: A média do aluno

Instrução de entrada

45

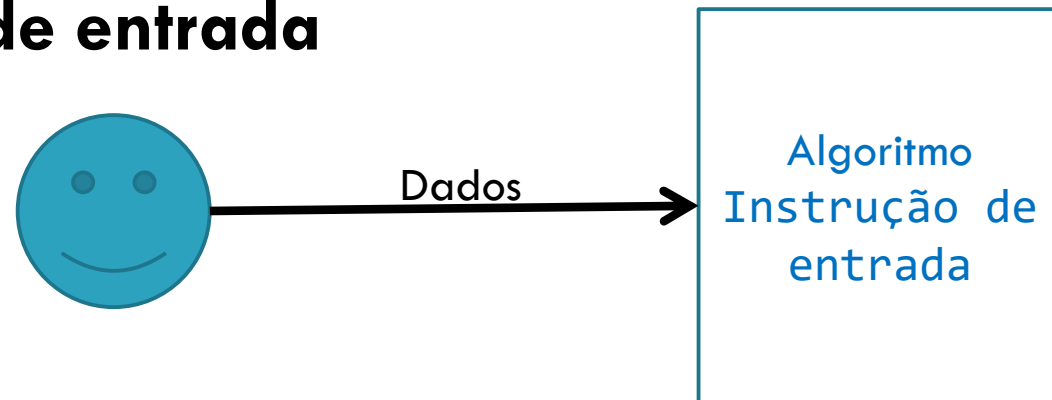
- ❑ Usuários enviam dados ao algoritmo por meio de **instruções de entrada**



Instrução de entrada

46

- ❑ Usuários enviam dados ao algoritmo por meio de **instruções de entrada**



- ❑ Na verdade, isso ocorre através de **periféricos ou dispositivos de entrada**

Periféricos de entrada

47



Profa. Simone Aires - Prof. Saulo Queiroz

Teclado: Periférico de entrada padrão

48

Dispositivo de entrada padrão:

Aquele de onde nossa instrução de entrada recebe os dados.

Default

Usuário humano



Dados



Entrada

Algoritmo
Instrução de
entrada

Periféricos de saída

49



Profa. Simone Aires - Prof. Saulo Queiroz

Monitor: Periférico de saída padrão

50

Dispositivo de saída padrão:

Aquele para onde nossa instrução de saída envia os dados.

Default

Usuário humano



Dados



saída

Algoritmo
Instrução de
saída

Instruções de Saída

Profa. Simone Aires - Prof. Saulo Queiroz

Instrução de saída:Linguagem C

52

- ▣ `printf("texto");`
- ▣ **Expandiremos essa sintaxe mais adiante no curso!**
- Exemplos:
 - ▣ `printf("Bom dia!");`
 - ▣ `printf("Bem vindo ao programa!");`
- Note que ambas instruções tem mesma sintaxe e fazem o mesmo, só muda o texto!

Instrução de saída: Erros de sintaxe

53

❑ printf("texto);



Instrução de saída: Erros de sintaxe

54

- ❑ `printf("texto);`
 - ❑ Faltou fechar as aspas

❑

❑

❑

❑

■

Instrução de saída: Erros de sintaxe

55

- ❑ `printf("texto);`
 - ❑ Faltou fechar as aspas
- ❑ `printf("texto")`
 - ❑
- ❑
 - ❑
 -

Instrução de saída: Erros de sintaxe

56

- ❑ `printf("texto);`
 - ❑ Faltou fechar as aspas
- ❑ `printf("texto")`
 - ❑ Faltou ponto-e-vírgula;
- ❑
 - ❑
 - ❑

Instrução de saída: Erros de sintaxe

57

- ❑ `printf("texto");`
 - ❑ Faltou fechar as aspas
- ❑ `printf("texto")`
 - ❑ Faltou ponto-e-vírgula;
- ❑ `Printf("texto");`
 - ❑

Instrução de saída: Erros de sintaxe

58

- ❑ `printf("texto);`
 - ▣ Faltou fechar as aspas
- ❑ `printf("texto")`
 - ▣ Faltou ponto-e-vírgula;
- ❑ `Printf("texto");`
 - ▣ Comando/instrução não reconhecido!
 - Seria printf?

Instrução de saída: Erros de sintaxe

59

□ printf “texto”;



Instrução de saída: Erros de sintaxe

60

- ❑ `printf “texto”;`
 - ❑ Comando requer parêntesis
- ❑ `print (“texto”);`

Instrução de saída: Erros de sintaxe

61

- ❑ `printf “texto”;`
 - ❑ Comando requer parêntesis
- ❑ `print (“texto”);`
 - ❑ Comando/instrução não reconhecido!

○ primeiro código-fonte

Profa. Simone Aires - Prof. Saulo Queiroz

○ programa “Olá Mundo” (*Hello World*)

63

- Tradição mundial como primeiro programa a se implementar no ensino de uma linguagem de programação
- Exercício: Faça um programa que imprime na tela a mensagem “Olá Mundo!”

Primeiro Programa

64

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

Biblioteca

```
int main() {  
    printf("Ola Mundo!");  
    return 0;  
}
```

Palavras
Reservadas

Comando de saída

Exercício

65

- ❑ Altere seu 1º programa imprimindo também na saída padrão seu nome. Apresente um comentário em bloco (antes do seu código) descrevendo os seguintes dados:
 - ❑ Autor do código, data de criação e email do autor
 - ❑ Explicação em uma frase
 - ❑ **Exercite sua atenção atentando rigorosamente à sintaxe discutida nos slides anteriores**

Primeiro Programa

66

```
#include <stdio.h>
/*
Autor: Simone Aires
Data: 01/03/2021
E-mail: sbkaminski@utfpr.edu.br
*/
int main() {
    printf("Ola Mundo!");
    printf("Simone!");
    return 0;
}
```

Caracteres de controle

67

- ❑ `\a` – alerta (beep)
- ❑ `\n` – nova linha
- ❑ `\r` – enter
- ❑ `\t` – tabulação (tab)
- ❑ `\b` – retrocesso
- ❑ `\“` – aspas
- ❑ `\\` - barra
- ❑ `\0` - nulo

Primeiro Programa

68

```
#include <stdio.h>
/*
Autor: Simone Aires
Data: 01/03/2021
E-mail: sbkaminski@utfpr.edu.br
*/
int main() {
    printf("Ola Mundo!");
    printf("\nSimone!");
    return 0;
}
```

Exercícios

69

1. Faça um programa que escreva a frase “Linguagem de Programacao Estruturada 2020-2” fazendo com que cada palavra seja impressa em uma linha separada.
2. Comente cada linha do código anterior explicando sua função.

Saída de Dados – função *printf*

Códigos de Formatação da Função *printf*

	Tipo	Exemplo
%c	Caracter	'a', 'k', 'p'
%d %i	Inteiro	10, -763
%f	Ponto flutuante – real	1.56, -9.3
%e	Notação científica	
%s	Cadeia de caracteres	"joaquim"
%p	hexadecimais	

Saída de Dados – função *printf*

- ❑ ***Criando um programa em C que escreva 3 frases com texto e valores***

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

```
void main ()
```

```
{
```

```
    printf ("O numero valido e :%d",2);
```

```
    printf("O valor em real e: %f",1.5);
```

```
    printf("A letra correta e : %c",'a');
```

```
}
```

Saída de Dados – função *printf*

Formatando casas decimais

```
#include<stdio.h>
void main ()
{
    printf("O preço e :%.2f", 1234.5681);
    printf("O preço e :%.3f", 1234.5681);
}
```

Saída

- preço e : 1234.56;
- preço e : 1234.568;

Palavras/Conceitos-chave

73

- ❑ Código-fonte e Linguagem C
- ❑ Sintaxe
 - ▣ Erro de Sintaxe/Erro de compilação,
 - ▣ Linguagem natural, Linguagem de programação, gramáticas, Noam Chomsky
 - ▣ *Case-sensitive*
 - ▣ Palavras reservadas
- ❑ Instrução
 - ▣ Comentários
 - ▣ Entrada e saída
 - ▣ Instrução, Periférico, Dispositivos padrão de entrada e de saída
 - ▣ Instrução de saída: `printf`

Instruções de Entrada

Profa. Simone Aires - Prof. Saulo Queiroz

Instrução de entrada

75

- Na instrução de **saída** quem decide qual a informação aparecerá na tela?



Instrução de entrada

76

- Na instrução de **saída** quem decide qual a informação aparecerá na tela?

- ▣ Em geral, somos nós, os programadores!

-

- ▣

Instrução de entrada

77

- Na instrução de **saída** quem decide qual a informação aparecerá na tela?
 - ▣ Em geral, somos nós, os programadores!
- Na instrução de entrada, quem decide a informação que será enviada ao algoritmo?
 - ▣

Instrução de entrada

78

- Na instrução de **saída** quem decide qual a informação aparecerá na tela?
 - ▣ Em geral, somos nós, os programadores!
- Na instrução de entrada, quem decide a informação que será enviada ao algoritmo?
 - ▣ O usuário do programa! Que conclusão tiramos?

Instrução de entrada

79

- O valor do dado de saída é conhecido durante a programação do código fonte pois somos nós que escrevemos.
 - ▣ Ex.: “Olá Mundo!”

Instrução de entrada

80

- O valor do dado de entrada não é conhecido durante o momento que estamos escrevendo o código. Ele pode ser qualquer coisa. Por exemplo:
 - ▣ Algoritmo para calcular o troco. Que valores preciso?
 -
 -

Instrução de entrada

81

- O valor do dado de entrada não é conhecido durante o momento que estamos escrevendo o código. Ele pode ser qualquer coisa. Por exemplo:
 - ▣ Algoritmo para calcular o troco. Que valores preciso?
 - o valor da nota dada como pagamento
 - o valor do preço do produto
 - ▣

Instrução de entrada

82

- O valor do dado de entrada não é conhecido durante o momento que estamos escrevendo o código. Ele pode ser qualquer coisa. Por exemplo:
 - ▣ Algoritmo para calcular o troco. Que valores preciso?
 - o valor da nota dada como pagamento
 - o valor do preço do produto
 - ▣ Depois é só mostrar a diferença entre os valores!

*Como vamos manusear/manipular em
nosso código-fonte um valor que só
será conhecido no futuro?*

***ou seja, só após o código ser compilado e o programa
executado? Note que esses valores podem mudar cada vez
que o programa for executado!***

84

Variáveis

Profa. Simone Aires - Prof. Saulo Queiroz

Variáveis

85

- ❑ É uma área da memória do computador onde nossos programas podem armazenar dados
- ❑ Antes de realizar uma leitura, precisamos criar uma variável em nosso código-fonte
- ❑ A instrução para criar uma variável é chamada **declaração de variável**

Declaração de variáveis

86

- Para declarar uma variável precisamos:
 1. Inventar um **nome** para a variável
 1. Ao longo do código, esse nome representará o valor informado pelo usuário
 2. Esse nome é chamado de **identificador** da variável
 2. Estabelecer a natureza ou **tipo do dado** que esperamos guardar dentro da variável

“Posso dar o nome que quiser à minha variável?”

Acho que já vi pergunta parecida
antes!!!!



“Posso dar o nome que quiser à minha variável?”

NÃO



Nomeação de Identificadores: Sintaxe

89

- ❑ Não podemos usar palavras reservadas
- ❑ Não podem conter caracteres especiais (exceto o caracter *underscore* _)
- ❑ Não podem iniciar com números
- ❑ Nome deve ter a ver com o que o código faz
 - ▣ Deve ser um **mnemônico**: aquilo que faz lembrar ...

Identificadores

90

- ❑ O padrão ANSI determina que podem ter qualquer tamanho, mas pelo menos os 31 primeiros devem ser significativos.
- ❑ Um nome pode ser maior que o número de caracteres significativos reconhecidos pelo compilador. Porém, os caracteres que ultrapassarem o limite serão ignorados.

Regras para Identificadores

91

- ❑ Maiúsculas e minúsculas são tratadas diferentemente. Ex: **conta**, **Conta**, **CONTA**.
- ❑ Não pode ser igual a uma palavra-chave de C.
- ❑ Não deve ter o mesmo nome que funções tanto as criadas pelo programador quanto as que estão na biblioteca C.

Tipos de dados

92

- Em geral, classificam-se em:
- **Numéricos:** usado para variáveis que armazenam números como, idade, valor de um produto, etc.
 - ▣ Na Linguagem C: `int`, `float`
- **Caracter:** usado para variáveis que armazenam letras e nomes
 - ▣ Na Linguagem C: `char`
- **Lógico:** usado para variáveis que armazenam valores lógicos falso ou verdadeiro.

Declaração de variável: Sintaxe

93

- ❑ <tipo_do_dado> <identificador>;
- ❑ Exemplos válidos:
 - ❑ `int ra; //inteiro`
 - ❑ `float valorProduto; //real`
 - ❑ `char nome_e_sobrenome; //caractere`

Declaração de variável: Sintaxe

94

- <tipo_do_dado> <identificador>;
- Exemplos **inválidos**: por que?
 - ▣ int 1ra;
 -
 - ▣ float valor Produto;
 -
 - ▣ char nome&sobrenome;
 -

Declaração de variável: Sintaxe

95

- `<tipo_do_dado> <identificador>;`
- Exemplos **inválidos**: por que?
 - ▣ `int 1ra;`
 - `//comeca com numero`
 - ▣ `float valor Produto;`
 - `//espaco em branco é caracter especial também`
 - ▣ `char nome&sobrenome;`
 - `//caracter especial &`

Tipos de variáveis em C

96

- C possui 5 tipos básicos:
 - ▣ **char, int, float, double e void**
- E 4 modificadores básicos:
 - ▣ **signed, unsigned, long e short**
 - ▣ Os 4 podem ser aplicados ao **int**
 - ▣ **long** pode ser aplicado ao **double**
 - ▣ **signed** e **unsigned** aplicados ao **char**

Modificadores de Variáveis

97

- São usados para alterar o significado de um tipo básico para adaptá-lo mais precisamente às necessidades de diversas situações.
- Controlam a maneira como as variáveis podem ser acessadas ou modificadas.

Tabela de Tipos Modificados

Tipo	Tamanho em bits	Faixa mínima
unsigned int	32	0 a 4.294.967.295
signed int	32	-2.147.483.647 a 2.147.483.647
short int	16	-32767 a 32767
long int	32 (no mínimo)	-2.147.483.647 a 2.147.483.647
signed long int	32 (no mínimo)	O mesmo que long int
unsigned long int	32 (no mínimo)	0 a 4.294.967.295
long double	80	Dez dígitos de precisão
long float	64	Mesmo significado de double

Códigos de Formatação



%c	Caracter
%d	Inteiro
%f	Ponto flutuante – real
%e	Notação científica
%s	Cadeia de caracteres
%x	Hexadecimais

Exemplo

- ▣ Faça um algoritmo para somar dois números.

```
# include <stdio.h>

void main ( )
{
    int X, Y, RES;
    X=4;
    Y=2;
    RES=X+Y;
    printf ("%d + %d = %d", X, Y, RES);
}
```

C:\Users\Simone\Documents\ex1.exe

4 + 2 = 6_

Qual o problema deste algoritmo ??

Constantes

101

- É tudo aquilo que é fixo ou estável.
- Uma constante pode ser declarada ou usada diretamente.
- Exemplos: `'\n'` (caractere), `"aline"` (string), **36** (inteiro)
- Definição de uma constante em C:
 - ▣ Existem 3 maneiras para criar constantes:
 - `#define [nome][valor]`
 - `[const] [tipo da variável][nome da variável]`
 - enumerations

Constantes

102

- **Define:**

Ex:

```
#define PI 3.14159265
```

- O identificador pode ter qualquer nome.
- Sempre que se necessitar do valor escreve-se o identificador, em vez do valor, até poderiam ocorrer enganos.
- Se for preciso alterar o valor no código, altera-se 1 vez, em vez de todas as vezes onde apareceria o valor.

Onde no código declaro as variáveis?

103

- Em qualquer momento antes de fazer uma leitura de um dado. Mas como se faz leitura?

Onde no código declaro as variáveis?

104

- ❑ Em qualquer momento antes de fazer uma leitura de um dado. Mas como se faz leitura?
- ❑ Pelo comando **scanf**

Comando de entrada (scanf)

- ☐ O valor informado pelo usuário será guardado (armazenado) na variável imediatamente após a instrução scanf.

Exemplo 1

```
int A;
```

```
scanf("%d", &A); //recebe o valor informado pelo usuário e armazena na variável A
```

Então agora podemos permitir que o usuário informe os números que devem ser somados.

Comando de entrada (scanf)

- ❑ O valor informado pelo usuário será guardado (armazenado) na variável imediatamente após a instrução scanf.

Exemplo 1

```
int A;
```

Código de formatação

```
scanf("%d", &A); //recebe o valor informado pelo usuário e armazena na variável A
```

Então agora podemos permitir que o usuário informe os números
que devem ser somados.

Comando de entrada (scanf)

- ❑ O valor informado pelo usuário será guardado (armazenado) na variável imediatamente após a instrução scanf.

Exemplo 1

```
int A;
```

Endereçamento de memória

```
scanf("%d", &A); //recebe o valor informado pelo usuário e armazena na variável A
```



Então agora podemos permitir que o usuário informe os números
que devem ser somados.

Comando de entrada (scanf)

- ❑ O valor informado pelo usuário será guardado (armazenado) na variável imediatamente após a instrução scanf.

Exemplo 1

```
int A;
```

Nome da variável

```
scanf("%d", &A); //recebe o valor informado pelo usuário e armazena na variável A
```



Então agora podemos permitir que o usuário informe os números
que devem ser somados.

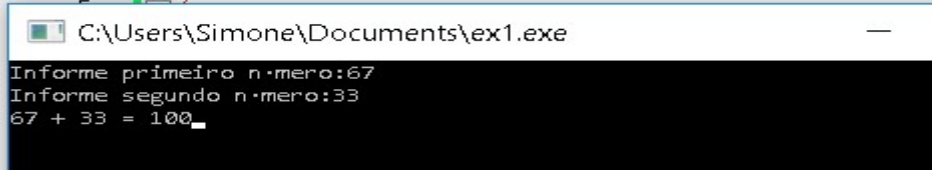
Exemplo

Faça um algoritmo para somar dois números informados pelo usuário.

```
# include <stdio.h>

void main ( )
{
    int X, Y, RES;
    printf("Informe primeiro número:");
    scanf("%d",&X);
    printf("Informe segundo número:");
    scanf("%d",&Y);
    RES=X+Y;
    → printf("%d + %d = %d", X, Y, RES);
}
```

Modifique o programa para calcular a
média entre os números!

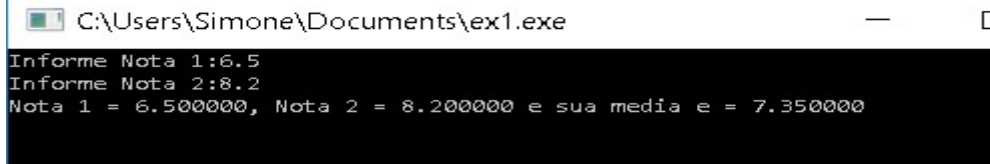


```
C:\Users\Simone\Documents\ex1.exe
Informe primeiro n.nero:67
Informe segundo n.nero:33
67 + 33 = 100_
```

Exemplo

Faça um programa em Linguagem C para calcular a média aritmética de duas notas informadas pelo usuário.

```
# include <stdio.h>
main ( )
{
    float N1, N2, MED;
    printf("Informe Nota 1:");
    scanf("%f",&N1);
    printf("Informe Nota 2:");
    scanf("%f",&N2);
    MED=(N1+N2)/2;
    printf("Nota 1 = %f, Nota 2 = %f e sua media e = %f", N1, N2,MED);
}
```



```
C:\Users\Simone\Documents\ex1.exe
Informe Nota 1:6.5
Informe Nota 2:8.2
Nota 1 = 6.500000, Nota 2 = 8.200000 e sua media e = 7.350000
```

Comando de Atribuição

Instrução de atribuição

- ❑ **Instrução** que permite ao programador alterar o valor de uma variável.
- ❑ Representada pelos símbolos \leftarrow ou $=$
- ❑ Sintaxe básica
 - ▣ `<identificador_da_variável> \leftarrow <valor>;`
 - ▣ `<identificador_da_variável> = <valor>;`
- ❑ O termo `<valor>` pode ser um(a):
 - ▣ Constantes: um valor digitado diretamente no código pelo programador
 - ▣ Variável
 - ▣ Expressão aritmética (estudaremos adiante)
 - ▣ Função (estudaremos mais tarde)

Instrução de atribuição

113

□ Atribuindo constante:

```
int var; //declarando variável  
var = 2; // atribuindo 2 à variável var
```

□ Atribuindo outra variável

```
int var, var2; //declarando variável  
var2 = -1; // atribuindo -1 à variável var  
var = var2; // atribuindo valor de var2 a var
```

Instrução de atribuição

114

- Atribuindo expressão aritmética:
 `int var; //declarando variável`
 `var = 2 + 5; // atribuindo 7 à variável var`
- Note que o próprio sistema avaliará a expressão aritmética para você e atribuirá o valor final à var

CUIDADO!

115

- ❑ O sinal de atribuição = **NÃO É UM SINAL DE COMPARAÇÃO**, como na matemática.
- ❑ Do **lado esquerdo** deve necessariamente haver uma variável e do **lado direito** um valor do tipo da variável!
O computador gravará o **valor** na **variável**
`float x; //declaração da variável x`
`x = 2.77; //atribuição de 2.77 à variável x`

Inicialização de variáveis

116

- Podemos usar a atribuição para inicializar variáveis
 - ▣ Juntamente com declaração. Ex.:
 - `int a = 0;`
 - ▣ **Logo** após inicializar. Ex.:
 - `int a;`
`a = 0;`
`a ← 0;`

Exemplo completo

117

```
/*  
    Programa que atribui e imprime o  
    valor aproximado da constante pi.  
    Saulo Queiroz  
*/  
void main()  
{  
    float pi;  
    pi = 3.14;  
    printf("O valor de pi é %f ", pi);  
}
```

Ilustração do funcionamento

118

```
/*
```

```
Programa que atribui e imprime o  
valor aproximado da constante pi.
```

```
Saulo Queiroz
```

```
*/
```

→ **void main()**

```
{
```

```
float pi;
```

```
pi = 3.14;
```

```
printf("O valor de pi é %f", pi);
```

```
}
```

Memória



Tela

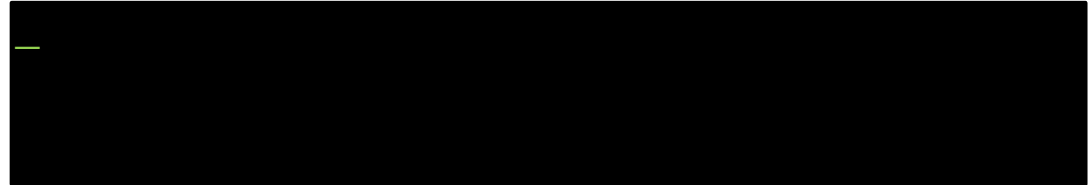


Ilustração do funcionamento

119

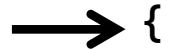
```
/*
```

```
Programa que atribui e imprime o  
valor aproximado da constante pi.
```

```
Saulo Queiroz
```

```
*/
```

```
void main()
```



```
{
```

```
float pi;
```

```
pi = 3.14;
```

```
printf("O valor de pi é %f", pi);
```

```
}
```

Memória



Tela

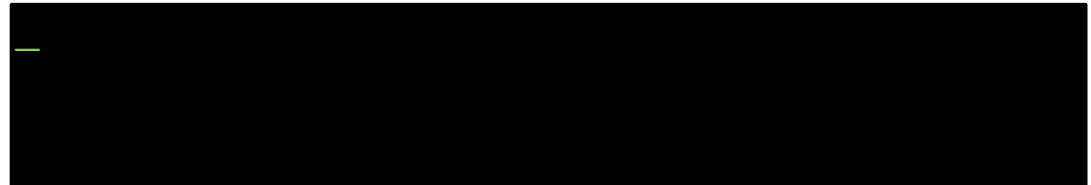


Ilustração do funcionamento

120

```
/*
```

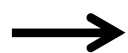
```
Programa que atribui e imprime o  
valor aproximado da constante pi.
```

```
Saulo Queiroz
```

```
*/
```

```
void main()
```

```
{
```



```
float pi;
```

```
pi = 3.14;
```

```
printf("O valor de pi é %f", pi);
```

```
}
```

Memória



Tela

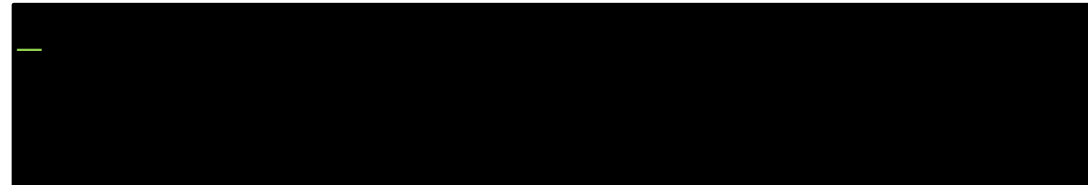


Ilustração do funcionamento

121

```
/*
```

```
Programa que atribui e imprime o  
valor aproximado da constante pi.
```

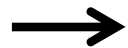
```
Saulo Queiroz
```

```
*/
```

```
void main()
```

```
{
```

```
float pi;
```



```
pi = 3.14;
```

```
printf("O valor de pi é %f", pi);
```

```
}
```

Memória



Tela

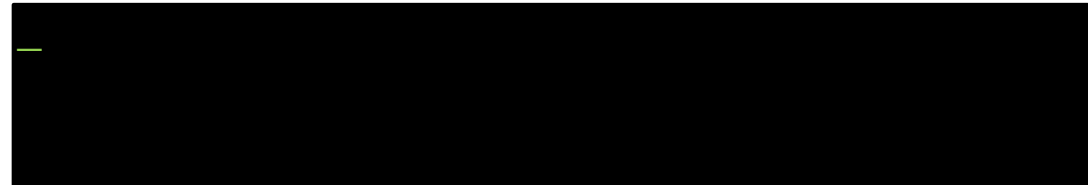


Ilustração do funcionamento

122

```
/*  
Programa que atribui e imprime o  
valor aproximado da constante pi.  
Saulo Queiroz  
*/  
void main()  
{  
    float pi;  
    pi = 3.14;  
    → printf("O valor de pi é %f", pi);  
}
```

Memória



Tela

O valor de pi é 3.14_

Ilustração do funcionamento

```
/*  
Programa que atribui e imprime o  
valor aproximado da constante pi.  
Saulo Queiroz  
*/  
void main()  
{  
    float pi;  
    pi = 3.14;  
    printf("O valor de pi é %f", pi);  
→ }
```

Memória



Tela

O valor de pi é 3.14_

Ilustração do funcionamento

124

```
/*  
 Programa que atribui e imprime o  
 valor aproximado da constante pi.  
 Saulo Queiroz  
*/  
void main()  
{  
    float pi;  
    pi = 3.14;  
    printf("O valor de pi é %f", pi);  
}
```

Memória

Tela

O valor de pi é 3.14_



Exercício em sala

125

- ❑ Faça um algoritmo que leia o valor de duas variáveis inteiras. Após isso seu código deve trocar os valores dessa variáveis.

Algoritmo Troca

126

// Programa que lê e troca o valor de duas variáveis Por. Saulo Queiroz

```
void main()
{
    int a=0, b=0;
    int aux; //variavel auxiliar
    printf("Informe um valor, depois o outro");
    scanf("%d",&a);
    scanf("%d",&b);
    aux = a;
    a = b;
    b = aux;
}
```

Profa. Simone Aires - Prof. Saulo Queiroz

Curiosidade

É possível resolver o problema da troca anteriormente enunciado utilizando menos de três variáveis? Por que? Quais as implicações?



Curiosidade: Discussão

128

- Vimos que cada variável corresponde a uma região específica de memória
- Nossos algoritmos assumem um computador no qual dois valores distintos não podem ocupar o mesmo lugar
 - ▣ Paradigmas de alternativas de computador tentam invalidar essa suposição! Ex.: computação quântica.

Biblioteca e operadores matemáticos

Operadores Matemáticos

130

Operador	Exemplo	Comentário
+	$x + y$	Soma x e y
-	$x - y$	Subtrai y de x
*	$x * y$	Multiplica x e y
/	x / y	Divide x por y
%	$x \% y$	Resto da divisão inteira de x por y
++	$x++$	Incrementa em 1 o valor de x
--	$x--$	Decrementa em 1 o valor de x

CUIDADO!

131

- ❑ OBS: o operador “/” (divisão) terá um resultado inteiro se os dois operandos forem inteiros. Para um resultado real, um dos dois operandos deve ser real (ou os dois).

- ❑ Exemplo:
 int x, y;
 float z, u, t;
 x = 2; y = 3; u = 3;
 z = x/y; // z terá o valor zero
 t = x/u; // t terá o valor 0.666667

Expressões

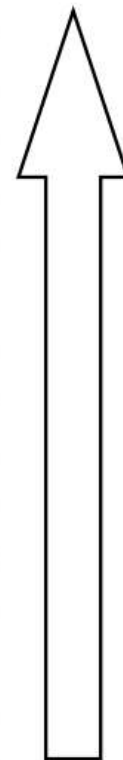
132

- Expressões são combinações de variáveis, constantes e operadores.
- Quando montamos expressões temos que levar em consideração a ordem com que os operadores são executados.
- Exemplo:
 $c = a*(b+d)/e;$

Precedência de Operadores

133

Operadores	Associatividade
() [] -> .	→
! ~ ++ --	←
* / %	→
+ -	→
<< >>	→
< <= > >=	→
== !=	→
&	→
&&	→
	→
^	→
	→
? :	←
= += -= *= /=	←



Maior prioridade

Menor prioridade

Precedência de Operadores

- A precedência entre operadores diz a prioridade que um operador tem em relação ao outro.
- Regra de precedências em C:
 - Operadores com o mesmo grau de precedência a prioridade é feita da esquerda para direita da expressão com a exceção dos operadores presentes nas linhas amarelas
 - Exemplos:
 - $2*3*4 = 6*4 = 24$
 - $2+3+4 = 5 + 4 = 9$
- Os parêntesis servem para alterar a ordem das precedências
 - Exemplos: $2*(3*4) = 2*12 = 24$
 - A multiplicação tem prioridade sobre a adição
 - Exemplo: $2+3*4 = ?$ ou seja

$2+(3*4)$	$2 + 3*4$
$2+12$	$5*4$
14 correto	20 errado

Operadores de Incremento e Decremento

- **Incremento: ++**

- ▣ Soma 1 ao seu operando

- ▣ Exemplo : `X++;` // Mesmo que : `x = x+1;`

- **Decremento: --**

- ▣ Subtrai 1 ao seu operando

- ▣ Exemplo : `X--;` // Mesmo que : `x = x-1;`

Prefixo ou sufixo

Prefixo : ++x

Sufixo : x++

- ▣ Tem diferença quando usado em expressão:

`x = 10;`

`y = ++x;` /* primeiro acrescenta 1 em x e depois

y é igual a 11 atribui a soma a y */

- ▣ Porém:

`x = 10;`

`y = x++;` /* primeiro atribui x a y e depois acrescenta

y é igual a 10 em x */

Biblioteca matemática

- C dispõe de algumas funções especiais para operações matemática.
- Para trabalhar com estas funções, deve-se usar em cada programa a biblioteca `math.h`.

```
#include <math.h>
```

- Existem diversas funções disponíveis como de potência, arredondamento, raíz quadrada, entre outros.
- Vamos abordar algumas funções...

Funções de Potência

Função pow()

- ❑ Retorna o valor da base elevada ao expoente, ou seja, calcula a exponenciação de um número.
- ❑ Recebe dois argumentos do tipo float, sendo respectivamente, base e expoente.
- ❑ Sintaxe:

`pow(base,expoente) → baseexpoente`

Exemplo:

$$3^2 \text{ pow}(3,2) = 9$$

$$2^{10} \text{ pow}(2,10) = 1024$$

Funções de Potência

Função sqrt()

- ❑ Função sqrt()
- ❑ Retorna o valor da raiz quadrada de um número, recebendo como argumento um float.
- ❑ Sintaxe: sqrt(num) num

Exemplo:

$$\sqrt{144} \rightarrow \text{sqrt}(144) \rightarrow (12 * 12 = 144)$$

Funções de Arredondamento

Função floor()

- ❑ Retorna o primeiro valor float, sem casas decimais, inferior ao número informado.
- ❑ Recebe um float como argumento.
- ❑ Sintaxe:

`floor(num) → num.casas`

Exemplo:

`3.2 → floor(3.2) → 3`

Funções de Arredondamento

Função ceil()

- ❑ Retorna o primeiro valor float, sem casas decimais, superior ao número informado.
- ❑ Recebe um float como argumento.
- ❑ Sintaxe:

$\text{ceil}(\text{num}) \rightarrow \text{num.casas} + 1$

Exemplo:

$3.2 \rightarrow \text{ceil}(3.2) \rightarrow 3 + 1 \rightarrow 4$

Funções Trigonométricas

Função `sin()`

- ❑ Retorna o valor do seno. Recebe como argumento o valor do tipo float em radianos.
- ❑ Obs.: $1 \text{ grau} = 0,017453 \text{ radianos}$
- ❑ Sintaxe:

`sin(num)`

Exemplo:

`sin(1000) → 0.826880`

Funções Trigonométricas

Função `cos()`

□ Sintaxe: `cos(num)`

Exemplo: `cos(1000)` → 0.532679

Função `tag()`

□ Sintaxe:

`tan(num)`

Exemplo:

`tan(1000)` → 1.470324

Funções Logarítmicas

Função `log()`

- ❑ Retorna o valor do logaritmo na base 2.
- ❑ Utiliza um argumento do tipo float.
- ❑ Sintaxe:

`log(num)`

Exemplo:

`log(10) → 1.000000`

Funções Logarítmicas

Função `log10()`

- ❑ Retorna o valor do logaritmo na base 10.
- ❑ Utiliza um argumento do tipo float.
- ❑ Sintaxe:

`log10(num)`

Exemplo:

`log10(10) → 2.302585`

Função - Resto inteiro da divisão

Função MOD

- ❑ Em linguagem C MOD é representado por %
- ❑ Utiliza um argumento do tipo int.
- ❑ Sintaxe:

num1 % num2

Exemplo:

$5 \% 2 \rightarrow 1$

Precisamos criar uma variável para armazenar o resultado dos cálculos de cada função.

Exemplo : `RESTO = 5%2;`

Para compilar com a biblioteca

gcc arquivo.c -o arquivo -lm

\\gera arquivo → ./arquivo

ou

gcc arquivos.c -lm

\\gera a.out → ./a.out

148

LISTA DE EXERCÍCIOS 01

Profa. Simone Aires - Prof. Saulo Queiroz

Conceitos-chave

149

- ❑ Memória
- ❑ Variável
- ❑ Declaração de variável
 - ▣ Identificador de variável, Mnemônico,
- ❑ Tipos de dados, número, caracter, lógico
 - ▣ inteiro, real, caracter, logico
- ❑ Instrução de entrada: leia
- ❑ Programa amigável, usabilidade