Fundamentos de Programação

Filipe Francisco

08 de março de 2018

Esquema de um programa em C

- Todo programa em uma linguagem tem um certo "esqueleto"
- Todo programa em C tamém tem algumas linhas essenciais
- "#include<stdio.h>"
  - "#include<...>": comando para incluir uma biblioteca
    - o sistema possui algumas, mas você também pode inserir a sua!
  - "stdio.h": do inglês, *standard input-output header*
  - inclui várias funções de entrada e saída predefinidas
    - exemplo de entrada: scanf
    - exemplo de saída: printf
- **int** main() {[código]}
  - espaço onde inserimos nosso código
  - importante: devemos ter um "**return** 0;" no código
    - ao chegarmos nesta linha, o programa é encerrado

Lendo em C

- Para ler do teclado o valor de uma variável, utilizamos scanf
- Ex: "scanf("%d",&x);"
  - comando para ler o valor de uma variável inteira x
  - "%d": indica que vamos receber um valor inteiro
  - "&": indica que vamos armazenar o valor lido em uma variável
  - "x": variável inteira na qual o valor será armazenado
- Podemos ler duas variáveis ao mesmo tempo? Sim!
  - ex: "scanf("%d %d",&a,&b);"
  - o primeiro valor inserido será atribuido à variável a, e o segundo, à variável b

Imprimindo em C

- Para imprimir algo na tela, utilizamos printf
- Podemos escrever uma mensagem acompanhada de caracteres especiais para indicar alguns dados a serem impressos
- Ex: "printf("A soma eh igual a %f\n",s);"
  - comando para imprimir uma mensagem acompanhada do valor de uma variável real (ou float) s
  - "%f": indica que queremos imprimir o valor de uma variável real
  - "\n": caractere especial para pular para uma nova linha
    - seria como "imprimir um enter"
  - "s": variável real a qual queremos imprimir o valor
    - como não estamos lendo o valor de uma variável, não usamos "&"

Alguns caracteres especiais

- imprimir '\': " \\"
- imprimir "'": " \" "
- imprimir "'": " \' "
- imprimir caractere nulo: " \0 "
- saltar para uma nova linha: " \n "
- imprimir uma tabulação: " \t "
- imprimir '%': " %% "
- para imprimir o valor de uma variável: depende do tipo da variável
  - " %d ", " %f ", " %c ", ...

⋮

Tipos primitivos de variáveis

- Variável: dado que pode receber um ou mais valores durante a execução do programa
- Inteiro (**int**)
  - número sem casas decimais
  - ex: 0, 42, -13, 12345, ...
  - representação para entrada/saída: "%d"
- Real (**float**)
  - número que permite casas decimais
  - ex: -1.23, 0.00001, 3.141592, 4, ...
    - obs: em C usamos ponto, não vírgula!
  - representação para entrada/saída: "%f"
    - imprimir um float mostra várias casas decimais
    - se quisermos limitar a uma certa quantidade de casas decimais, podemos usar "%.#f"
    - ex: "%.2f"

Alguns operadores

- '=': operador de atribuição
  - ex: "a=0;"
- '+': operador de soma
  - ex: "a=b+c;"
- '-': operador de subtração
  - ex: "a=b-c;"
  - obs: com "a=-b;", fazemos com que a receba o valor inverso de b!
- '*': operador de multiplicação
  - ex: "a=b*c;"

Alguns operadores

- '/': operador de divisão
  - ex: "a=b/c;"
  - obs: se a é uma variável inteira, a atribuição acima removerá todas as casas decimais
  - obs: se ambos b e c forem variáveis inteiras, a divisão resultará em um valor inteiro!
  - como resolver? com o operador **cast**!
- '%': operador de módulo (resto da divisão)
  - ex: "a=b%c;"
  - obs: só trabalha com inteiros!

Alguns operadores

- cast: operador de molde
  - força a variável imediatamente seguinte a ele, a ser tratada como de outro tipo
  - útil, por exemplo, para resolver problemas nas divisões com dois valores inteiros
  - ex: "a=3/2;" será igual a 1, independente se a é inteiro ou real
  - para resolver isto, aplicamos um cast para real sobre um dos valores da divisão
    - se pelo menos uma das variáveis é real, o compilador trabalhará a divisão como real, e não como inteiro!
  - para "a=(float)3/2;" ou "a=3/(float)2;", temos a=1.5!

Precedência de operadores

- Em uma fórmula do tipo "a=1*2+4/(5-3);", o que devemos fazer primeiro?
- Quando estudamos matemática, vimos que resolvemos primeiros os parêntes, depois multiplicação e divisão, e depois adição e subtração
- Esta ideia nos remete à precedência dos operadores
- A precedência indica ao compilador que operações devem ser executadas primeiro
- Ordem de precedência:
  - parênteses
  - cast
  - multiplicação, divisão e módulo
  - soma e subtração

Outros operadores

- '++': operador de incremento por 1
  - ex: "a++;"
  - é o mesmo que escrever "a=a+1;"
- '−−': operador de decremento por 1
  - ex: "a−−;"
  - é o mesmo que escrever "a=a-1;"

Outros operadores

- '+=': operador de incremento por algum valor
  - ex: "a+=b;"
  - é o mesmo que escrever "a=a+b;"
- '-=': operador de decremento por algum valor
  - ex: "a-=b;"
  - é o mesmo que escrever "a=a-b;"
- '*=': operador de multiplicação por algum valor
  - ex: "a*=b;"
  - é o mesmo que escrever "a=a*b;"
- '/=': operador de multiplicação por algum valor
  - ex: "a/=b;"
  - é o mesmo que escrever "a=a/b;"
- '%=': operador de multiplicação por algum valor
  - ex: "a%=b;"
  - é o mesmo que escrever "a=a%b;"
  - obs: novamente, módulo só funciona com números inteiros!

Criando e acessando variáveis

- Como criar uma variável?
  - basta inserirmos o tipo de variável e o seu identificador
    - ex: "**int** n;"
  - podemos aproveitar uma mesma linha para criar várias variáveis de mesmo tipo
    - ex: "**float** x, y, z;"
  - não podemos criar duas variáveis com o mesmo identificador!
- Como atribuir valor a uma variável?
  - utilizamos o '=' para atribuir valor!
    - ex: "x=2;"
  - também podemos atribuir valor a uma variável no momento da sua criação
    - ex: "**int** i=0;"

Tipos primitivos de variáveis (continuação)

- Caractere (**char**)
  - representa um caractere (letra ou símbolo)
  - ex: 'a', 'C', '$', '.' ...
    - obs: o C é case sensitive! (a $\neq$ A)
  - representação para entrada/saída: "%c"
- Booleano (**?**)
  - representa um valor booleano
  - só pode assumir dois valores: verdadeiro ou falso!

Booleano em C

- C não tem uma representação fixa para booleanos
  - não há uma representação do tipo true/false
- Então, como trabalhar com a ideia de verdadeiro e falso em C?
- Existem bibliotecas para trabalhar com booleano, mas o próprio C trabalha isso de uma maneira simples!
- Em C, vamos utilizar uma variável inteira, com 0 representando falso e 1 representando verdadeiro!
  - apesar disso, o compilador trata 0 como falso e qualquer outro valor como verdadeiro (não apenas o 1!)
- E qual a utilidade de booleano na nossa vida?

Condicionais

- Variáveis booleanas são essenciais para **condicionais**!
- Até aqui, só vimos códigos que lêem valores, efetuam operações matemáticas e imprimem algo
- A partir daqui, começamos a incrementar nossos algoritmos
- O que chamamos de condicional?
  - é uma maneira de verificar uma condição
  - a partir do resultado desta verificação, efetuamos uma certa tarefa
- Exemplo: como verificar se um número é par ou ímpar?

Condicionais

Algoritmo: Par ou Impar?

```
1  inicio
2      inteiro a, b
3      leia a
4      b = a%2
5      se b = 0 então
6          imprima "O número é par!"
7      fim
8      senão
9          imprima "O número é ímpar!"
10     fim
11 fim
```

Condicionais

Algoritmo: Par ou Impar?

```
1  inicio
2      inteiro a, b
3      leia a
4      b = a%2
5      se b = 0 então
6          imprima "O número é par!"
7      fim
8      senão
9          imprima "O número é ímpar!"
10     fim
11 fim
```

- Problema: '=' é o operador de atribuição!
  - ao fazermos "b=0", estamos atribuindo o valor 0 a b!
- Em vez disso, queremos comparar se b é igual a 0!
- Para fazer esta comparação, utilizaremos um outro operador!

Condicionais

Algoritmo: Par ou Impar?

```
1  inicio
2      inteiro a, b
3      leia a
4      b = a%2
5      se b == 0 então
6          imprima "O número é par!"
7      fim
8      senão
9          imprima "O número é ímpar!"
10     fim
11 fim
```

- '==': operador de igualdade
- Checa se dois valores são iguais
- A resposta seria verdadeiro ou falso, porém C não tem booleano!
- Em C, o operador '==' é igual a 1 se os dois valores forem iguais, e 0 se forem diferentes!

Condicionais

Algoritmo: Par ou Impar?

```
1  inicio
2      inteiro a
3      leia a
4      se a%2 == 0 então
5          imprima "O número é par!"
6      fim
7      senão
8          imprima "O número é ímpar!"
9      fim
10 fim
```

- Detalhe: não precisamos criar uma nova variável apenas para armazenar o módulo!
- Podemos fazer tudo em uma única linha!

Condicionais

- Sintaxe de um condicional simples
  - **se** [condição] **então** [código]
  - verifica se uma dada condição é verdadeira ou falsa
  - se a condição é verdadeira, executa as linhas de código
- A condição normalmente nos leva a alguma sentença lógica, algo que possa ser avaliado como verdadeiro ou falso
- Ou seja, remete diretamente à lógica de Matemática Básica!

Condicionais

- Podemos incrementar um pouco mais e fazer um condicional mais completo
  - **se** [condição] **então** [código 1] **senão** [código 2]
  - verifica se uma dada condição é verdadeira ou falsa
  - se a condição é verdadeira, executa as linhas do código 1
  - se a condição é falsa, executa as linhas do código 2
- Apesar de ser o único exemplo visto hoje, a igualdade não é a única condição que podemos verificar
- Veremos mais condições na próxima aula!