

Programação de Computadores

Profs. Marilton Sanchotene de Aguiar e Giovani Parente Farias
`{marilton, giovani.pfarias}@inf.ufpel.edu.br`

Cursos de Ciência e Engenharia de Computação
Centro de Desenvolvimento Tecnológico
Universidade Federal de Pelotas

2020/2



Semana I

Conceitos Básicos



Introdução

- Criada por Denis Ritchie em 1972 (Bell Labs, USA), falecido em 2011 (veja mais em <http://bit.ly/2Mj0VvP>).
- Sua primeira utilização importante foi a escrita do S.O. Unix
- Em meados de 1970 ela foi liberada para uso em Universidades
- Em 1980 já existiam várias versões de compiladores de diversas empresas (não restritos ao ambiente Unix)



Veja

Assista os vídeos da playlist disponível em http://bit.ly/pc_antes

A Tarefa de Programar

É uma tarefa intelectual.

Envolve:

- Traduzir especificações em código
- Analisar código escrito por outros programadores
- Modificar código para remover defeitos
- Manipulação direta de textos escritos em linguagens de programação (linguagens não-naturais ou não-amigáveis)

O desconhecimento dos recursos da linguagem e de técnicas de codificação aumenta o esforço da programação.



- Envolvem a escrita e organização (separação em módulos, endentação, etc) de textos carregados de símbolos
- Inclusão de informações necessárias a futuras atividades de manutenção (comentários)
- Inclusão de dispositivos de segurança no código



São recomendações genéricas aplicáveis a muitas linguagens de programação. As diretrizes abordam:

- Dados
- Nomes
- Estrutura de controle
- Leiaute
- Comentários



- Programação defensiva: tudo pode vir a dar errado
- Testar erros nas entradas e tentar se comportar bem quando os encontra
- Aumentar o número de verificações no código
 - Validar entrada de dados via interface do usuário
 - Validar dados fornecidos de uma rotina a outra
 - Consistência em estruturas de dados



Principais Características

- C é uma linguagem de propósito geral, adequada à programação estruturada e juntamente com Fortran, é a principal linguagem para uso científico:
 - Compiladores
 - analisadores léxicos
 - bancos de dados
 - editores de texto...
- Portabilidade
- Modularidade
- Compilação separada
- Recursos de baixo nível (tratamento de caracteres)
- Geração de código eficiente
- Confiabilidade
- Regularidade
- Simplicidade
- Facilidade de uso



- São as regras para cada construção válida em uma linguagem específica. Estão relacionadas com:
 - Tipos: definem as propriedades dos dados
 - Declarações: expressam as partes de um programa
 - Funções: especificam as ações que um programa executa, quando roda
- Determinação e alteração de valores
- Chamada de funções de I/O



Funções

- São os blocos básicos dos programas em C
- Há funções básicas, definidas na biblioteca C:
`printf()`, `scanf()`, `getchar()`, `putchar()`
- Funções podem ser definidas pelo programador
- Todo programa C inicia sua execução chamando a função `main()`, sendo obrigatório a sua declaração no programa principal.

Veja

- exemplo-01.c

Identificadores

- São os nomes usados para fazer referência a
 - Variáveis
 - Funções
 - Módulos
 - Rótulos
 - Demais objetos definidos pelo usuário
- Regras de Formação:
 - O primeiro caractere deve ser uma letra ou sublinhado
 - Os 32 primeiros caracteres são significativos
 - *Case sensitive* - difere maiúsculas de minúscula

Exemplos

a, A, Nota, nota, _nota

- Declarar sempre todas as variáveis
- As declarações devem ser acompanhadas de comentários descritivos
 - Cada declaração deve ocupar uma linha
 - Declaração próximo do uso (sempre que possível)
 - Os tipos e nomes das variáveis devem ser alinhados em colunas
 - Inicializar todas as variáveis em sua declaração
 - Documentar porque da inicialização com um dado valor, se não for óbvio



Escopo

- Minimizar o escopo das variáveis: preferir variáveis locais a uma rotina do que locais a um módulo, que são preferíveis a variáveis globais
- Minimizar o uso de variáveis globais: variáveis globais causam problemas, dificulta reuso, quebra de modularidade, etc.
- Usar variáveis globais somente para:
 - Preservação de valores globais ao programa inteiro
 - Simulação de constantes com nome
 - Dados de uso extremamente comum
 - Dados que são passados a rotinas de nível profundo sem serem manipulados por rotinas de nível intermediário

Finalidade

- Usar cada variável com uma única finalidade
 - Não reutilizar nomes com propósitos diferentes
 - Evitar variáveis com dois significados
 - Conferir se todas as variáveis declaradas são usadas

Nomes

- Todos os nomes devem descrever de forma completa e precisa a entidade que representam
 - Usar nomes auto-explicativos (auto-documentáveis) para as variáveis, rotinas e todos os tipos de elementos da linguagem
 - Os próprios nomes contém informação sobre o funcionamento do código
 - Usar nomes mais próximos do problema do que da implementação

Nomes

```
int x, y, z, x1, x2;
```

- Problemas de compreensão causados:
 - É impossível a outro programador deduzir o propósito de cada variável
 - Mesmo o programador que escreveu o código pode ter problemas para entendê-lo após alguns dias
 - A atividade de debug torna-se mais difícil



Declaração

- Todos os identificadores devem representar palavras ou frases do idioma utilizado
- Código para distribuição internacional: usar inglês
- Se for usado português, considere:
 - Usar nomes de funções no infinitivo, imperativo ou indicativo presente
 - Não omitir preposições
- Evitar abreviações: Abreviações devem ser usadas quando as linguagens não permitem nomes longos
- Os nomes devem ser pronunciáveis
- Os nomes devem diferenciáveis pelo seus primeiros caracteres: Evitar diferenciar por numeração (ex: temp, temp2)
- Evitar a ocorrência de identificadores com nomes que possam ser confundidos



Diretrizes de denominação

Índices: usar os tradicionais i, j, k apenas em casos simples. Prefira nomes mais significativos (ex: contadorDeRegistros)

Variáveis de Status: prefira nomes que indiquem o significado (ex: dadoPronto, tipoDeRelatório)

Variáveis Temporárias: nomes com significado. Evitar o uso de temp

Variáveis booleanas: usar nomes significativos como feito, erro, achou, sucesso. Evitar nomes que não indiquem valor lógico ou nomes negativos (ex: nãoAchou)

Tipos enumerados: usar prefixos comuns para nomes pertencentes ao mesmo grupo (corAzul, corVermelha, corVerde)

Constantes: usar nomes indicativos de significado da constante (ex: totalDeCiclos)



- O Tipo serve (grosso modo) para determinar
 - Como os valores serão armazenados
 - Quais operações poderão ser realizadas sobre estes valores
- Tipos para Dados e para Funções
 - Tipos escalares
 - Tipos não-escalares: tipo de estrutura, tipo de união, tipo de matriz



- Tipos Aritméticos

Tipos Inteiros: char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long

Tipos Flutuantes: float, double, long double

- Tipos Ponteiros

- para funções
- para objetos de dados
- para tipos incompletos



Tipos Inteiros

Tipo	Tamanho (bytes)	Intervalo
char	1	-128 a 127
unsigned char	1	0 a 255
signed char	1	-128 a 127
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
short	2	-32.768 a 32.767
unsigned short	2	0 a 65.535
long	4	-2.147.483.648 a 2.147.483.647
unsigned long	4	0 a 4.294.967.295



Tipos Flutuantes

Tipo	Tamanho (bytes)	Intervalo
float	4	1.2E-38 a 3.4E+38
double	8	2.3E-308 a 1.7E+308
long double	16	3.4E-4932 a 1.1E+4932

http://en.wikipedia.org/wiki/C_data_types

Veja

- exemplo-02.c
- exemplo-03.c



Diretrizes para Tipos Numéricos

- Evitar os chamados números mágicos, usando constantes no lugar de literais (exceto no caso de 0, 1)
- Usar constantes ao invés de valores explícitos no código

```
#define Pi 3.141592
```
- Verificar se o valor dos divisores não pode ser 0
- Explicitar todas as conversões de tipo
- Inteiros
 - Tratar a possibilidade de truncamento
 - Tratar a possibilidade de estouro (overflow), inclusive em resultados intermediários
- Ponto-flutuante
 - Evitar somas e subtrações de números de magnitude muito diferentes
 - Tratar possibilidades de overflow e underflow
 - Tratar possibilidades de erro de arredondamento



Operadores

- Atribuição: =
- Aritméticos: +, -, *, /, %
- Relacionais: >, \geq , <, \leq , =, \neq
- Lógicos: &&, ||, !
- Outros operadores:
 - $i += 2; (i = i+2;)$
 - $x *= y+1; (x = x*(y+1);)$
 - $d -= 3; (d = d-3;)$

Atenção

- Em C, falso é 0 (zero). Qualquer outro valor é considerado verdadeiro (inclusive negativo)
- Operadores relacionais e lógicos têm menor precedência do que operadores aritméticos

- `++x` (incrementa `x` antes de usar o seu valor)
- `x++` (incrementa `x` depois de ser usado)
- `--y` (decrementa `y` antes de usar o seu valor)
- `y--` (decrementa `y` depois de ser usado)



Operadores Bit a Bit

Operador	Ação
&	AND
	OR
^	XOR
~	Complemento
<<	Shift à esquerda
>>	Shift à direita

$$11000001 \& 01111111 = 01000001$$

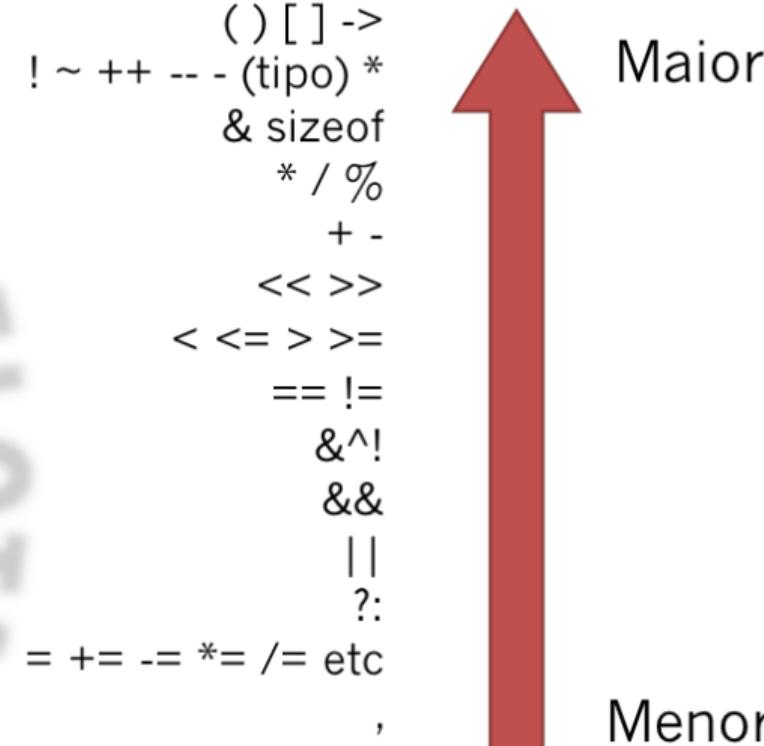
$$10000000 | 00000011 = 10000011$$

$$01111111 \wedge 01111000 = 00000111$$

Veja

- exemplo-04.c

Precedência



printf()

- Sintaxe:

```
printf("expressao de controle", argumentos);
```

- É uma função de E/S que permite escrever no dispositivo padrão (tela)
- A expressão de controle pode conter:
 - caracteres que serão exibidos na tela
 - códigos de formatação
- Cada argumento deve ser separado por vírgulas



printf()

- \n (nova linha)
- \t (tabulação)
- \b (retrocesso)
- \"(aspas)
- \\(barra)
- \f (salta formulário)
- \0 (nulo)
- %c (caractere)
- %d (decimal)
- %e (notação científica)
- %f (ponto flutuante)
- %o (octal)
- %s (string)
- %u (decimal sem sinal)
- %x (hexadecimal)

Veja

- exemplo-05.c
- exemplo-06.c

scanf()

- Sintaxe:

```
scanf("expressao de controle", argumentos);
```

- É uma função de E/S que permite ler dados formatados da entrada padrão (teclado)
- A lista de argumentos deve consistir dos endereços das variáveis na memória do computador.
- Como obter o endereço de uma variável??
 - Colocando um & junto ao nome da variável.
 - Ex.: &a (fornece o endereço da variável a)

Veja

- exemplo-07.c
- Assista os vídeos da playlist disponível em http://bit.ly/pc_intro