

Introdução à Linguagem C

- A linguagem C surgiu na década de 1970 e foi inventada por Dennis Ritchie, rodando o sistema operacional Unix.
- Linguagem que pode ser usada em programação de baixo e alto nível.
- Linguagem de programação genérica, o que permite criar diversos tipos de aplicações.
- Existem diversas implementações da linguagem C.
- É possível utilizar diversos ambientes de desenvolvimento, como o Eclipse, ... [colocar os outros ambientes de programação]

Características da linguagem

- A linguagem C é *case sensitive*.
- A linguagem C é compilada.
- exemplo01.c
- Estrutura geral de um programa em C.
- A linguagem C possui um conjunto de palavras-reservadas [colocar a tabela com as palavras-reservadas].

Tipos de dados

- A linguagem C possui 5 tipos de dados: char, int, float, void, double.
- Caracteres
 - char
 - Representa caracteres ou valores numéricos.
 - Tamanho da representação é de 1 byte ou 8 bits de informação.
- Inteiros
 - int
 - Representa 2 ou 4 bytes.
- Números em ponto flutuante (reais)
 - float
 - double (o dobro da precisão do tipo float)
- Void é um tipo especial e serve para indicar que o tipo é vazio.
- Existem quatro tipos de modificadores para os tipos: signed, unsigned, long e short.

Tipo	Num de bits	Formato para leitura com scanf	Intervalo	
			Início	Fim
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	16	%i	-32.768	32.767
unsigned int	16	%u	0	65.535
signed int	16	%i	-32.768	32.767
short int	16	%hi	-32.768	32.767
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi	-32.768	32.767
long int	32	%li	-2.147.483.648	2.147.483.647
signed long int	32	%li	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	0	4.294.967.295
float	32	%f	3,4E-38	3,4E+38
double	64	%lf	1,7E-308	1,7E+308
long double	80	%Lf	3,4E-4932	3,4E+4932

Nomes de variáveis

- Regras para criação dos nomes das variáveis:
 - O nome das variáveis deve começar com uma letra ou um sublinhado (_).
 - O nome da variável não pode ser igual a uma das palavras-reservadas e nem igual ao nome de uma função declarada pelo programador ou pelas bibliotecas C.

Declaração e inicialização de variáveis

- Forma padrão de declarar variáveis: <tipo-variável> <nome-ou-lista-variáveis>;
- As variáveis podem ser declaradas como locais, globais ou na lista de parâmetros.
- É possível inicializar a variável no momento da sua declaração.

Constantes

- São valores que não são alterados ao longo da execução dos programas.
- Para indicar que um número está numa base diferente da decimal, utilize 0X para indicar que o número é hexadecimal e 0 para indicar que está na base octal.

Operadores aritméticos

- Relação dos principais operadores aritméticos em C:

Operador	Ação
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto de divisão (de inteiros)
++	Incremento (inteiro e ponto flutuante)
--	Decremento (inteiro e ponto flutuante)

- Os operadores ++ e -- podem ser pré-fixados e pós-fixados e existe diferença entre eles. Os operadores pré-fixados incrementam e depois retornam o valor da variável já incrementada. Os operadores pós-fixados retornam o valor da variável sem o incremento e depois incrementam a variável.
 - Exemplo: x=23; y=x++; e x=23; y=++x;
 - No primeiro teremos y igual a 23 e x igual a 24. No segundo teremos y igual a 24 e x igual a 24.
- O operador = é utilizado para atribuição de valores.

Operadores relacionais e lógicos

- Relação dos principais operadores relacionais em C:

Operador	Ação
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

- Os operadores relacionais retornam verdadeiro (1) e falso (0).

- Relação dos principais operadores lógicos em C:

Operador	Ação
&&	AND (E)
	OR (OU)
!	NOT (NÃO)

Comandos de entrada e saída de dados

- Leitura e escrita de caracteres
 - O comando `getchar()` é utilizado para ler um caracter a partir do teclado.
 - Protótipo do comando: `int getchar(void)`
 - O comando `putchar()` é utilizado para escrever um caracter na tela.
 - Protótipo do comando: `int putchar(int c)`
- Leitura e escrita de strings
 - O comando `gets` é utilizado para ler uma string a partir do teclado.
 - Protótipo: `char *gets (char *s)`
 - O comando `puts` é utilizado para escrever uma string na tela.
 - Protótipo: `int puts (char *s)`
- exemplo-ler-escrever-string.c
- Escrita de valores
 - O comando `printf` é utilizado para imprimir valores na tela.
 - Protótipo: `int printf (char *str, ...)`
 - Relação de possíveis formatos:

Código	Formato
%c	Um caracter (char)
%d	Um número inteiro decimal (int)
%i	O mesmo que %d
%e	Número em notação científica com o "e"minúsculo
%E	Número em notação científica com o "e"maiúsculo
%f	Ponto flutuante decimal
%g	Escolhe automaticamente o melhor entre %f e %e
%G	Escolhe automaticamente o melhor entre %f e %E
%o	Número octal
%s	String
%u	Decimal "unsigned" (sem sinal)
%x	Hexadecimal com letras minúsculas
%X	Hexadecimal com letras maiúsculas
%%	Imprime um %
%p	Ponteiro

- É possível também indicar o número de casas decimais de um número em ponto flutuante.
Exemplo: `%10.4f`

- Leitura de valores
 - O comando scanf é utilizado para a leitura de valores a partir do teclado.
 - Protótipo: int scanf (char *str, ...);
 - Necessidade do operador & para as variáveis que não sejam ponteiros.
 - Relação dos formatos possíveis de leitura:

Código	Formato
%c	Um único carácter (char)
%d	Um número decimal (int)
%i	Um número inteiro
%hi	Um short int
%li	Um long int
%e	Um ponto flutuante
%f	Um ponto flutuante
%lf	Um double
%h	Inteiro curto
%o	Número octal
%s	String
%x	Número hexadecimal
%p	Ponteiro

- exemplo-printf-scanf.c

Comandos condicionais

- Os comandos condicionais são comandos que avaliam expressões e tomam decisões a respeito do que deve ser executado a partir de uma avaliação feita.
- Comando if...else
 - Não há necessidade de chaves se houver um único comando.
 - exemplo-maior-valor.c
 - Uma outra opção é usar o comando ?
 - Sintaxe: (expressão) ? (operação verdadeira) : (operação falsa)
- Comando switch...case
 - É outro comando de controle de fluxo.
 - Sintaxe:


```
switch (expressão) {
    case x:
        lista de operações;
        break;
    case y:
        lista de operações;
        break;
    default:
        lista de operações;
}
```
 - O comando default e break não são obrigatórios.
 - exemplo-switch-case.c

Comandos de laços (loop)

- A linguagem C possui três comandos de laços de repetição: for, while e do...while.
- Comando for
 - Sintaxe:
for (inicialização; critério de parada; incremento ou decremento) {
 lista de operações;
}
 - exemplo-comando-for.c
 - Exemplo de uso do comando for:
 - for (i=0;i<1000000;i++);
 - É utilizado para fazer atrasos nos programas.
- Comando while
 - Diferentemente do comando for, o comando while não apresenta condição inicial e nem controle de incremento/decremento.
 - Sintaxe:
while (condição) {
 lista de operações;
}
 - exemplo-comando-while.c
- Comando do-while
 - Sintaxe:
do {
 lista de operações;
} while (condição);
 - exemplo-comando-do-while.c
- Comando continue
 - O comando é utilizado dentro de laços de repetição para finalizar apenas a execução da iteração, pulando todos os demais comandos dele.
 - exemplo-comando-continue.c
- Comando break
 - O comando pode ser utilizado dentro de laços de repetição para finalizar a execução do laço, pulando todos os demais comandos e não considerando a própria condição de parada.
 - exemplo-comando-break.c

Operações matemáticas

- As operações matemáticas mais avançadas são estabelecidas dentro da biblioteca math.h.
- Assim, sempre que necessitar dessas operações, deve-se declarar no início do programa.

#include <math.h>

- Entre as funções matemáticas, é possível utilizar operações trigonométricas, hiperbólicas, exponenciais, logarítmicas, entre outras.
- Função de potência
 - A função pow realiza a exponenciação de um número por outro.
 - Esta função retorna um valor tipo double e seus argumentos também são double.
 - A sua sintaxe é: pow (base, expoente).
- Função de raiz quadrada
 - Embora seja possível calcular a raiz quadrada de um número, utilizando o expoente com o valor 0,5, a linguagem C possui uma função própria, que é o comando sqrt ().
 - A sua sintaxe é: sqrt (valor)
 - O valor passado também deve ser do tipo double.

- Função de arredondamento
 - A função de arredondamento é `ceil ()`.
 - A sua sintaxe é: `ceil (valor)`
 - Esta função recebe um valor do tipo `double` como parâmetro e retorna um valor arredondado. Por exemplo, se `x = ceil (3.2)`, o valor de `x` passará a ser 4.
- Função de truncamento
 - A função de truncamento de números em C é o comando `floor ()`.
 - A sua sintaxe é: `floor (valor)`
 - Esta função recebe um valor do tipo `double` como parâmetro e retorna um valor truncado. Por exemplo, se `x = floor (3.2)`, o valor de `x` passará a ser 3.
- exemplo-funcoes-matematicas.c
- Função de número
 - A linguagem C possui duas funções para calcular o valor absoluto de um número. A função `abs (valor)` e a função `fabs (valor)`. A diferença é que a primeira é utilizada para calcular valores absolutos do tipo `double` e a segunda é utilizada para calcular valores absolutos para o tipo inteiro.
- Funções para geração de números aleatórios
 - Existem situações em que é necessário gerar números aleatórios.
 - A linguagem C possui o comando `rand()` para gerar números aleatórios. A função retorna um valor inteiro entre 0 e `RAND_MAX`, em que `RAND_MAX` é o maior valor que pode ser obtido por essa função. Este valor é definido em `stdlib.h`.
 - Quando se deseja limitar o valor máximo de `rand`, utiliza-se a seguinte declaração: `rand() % k`; com `k` sendo um número inteiro que determina o valor teto para os números gerados.
 - exemplo-numero-randomicos.c

Ponteiros

- Ponteiros são tipos especiais em C que servem para armazenar endereços de memória.
- Os ponteiros também possuem um tipo, pois ao armazenar o endereço, é preciso identificar qual é o tipo de valor que será armazenado.
- Sintaxe: `<tipo-ponteiro> *<nome-variável>;`
- O asterisco indica ao compilador que a variável não irá guardar um valor, mas apenas um endereço especificado.
- É possível também descobrir o valor do endereço de uma variável. Para isso, basta usar o operador `&` antes da variável para pegar o endereço.
- Trecho de exemplo:

```
int valor = 10;
int *pt;
pt = &valor;
```

- exemplo-ponteiro-01.c
- É possível também realizar operações de incremento e decremento com os ponteiros. Quando incrementamos um ponteiro ele passa a apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta. Ou seja, se temos um ponteiro para um inteiro e o incrementamos ele passa a apontar para o próximo inteiro. Esta é a razão pela qual o compilador precisa saber o tipo de um ponteiro.
- A operação de decremento funciona de forma semelhante, mas com a diminuição dos endereços.
- Exemplo de incremento de ponteiro: `p++`
- Vale dizer que `p++` é diferente de `(*p)++`. O primeiro representa um incremento no endereço do ponteiro `p` e o segundo representa um aumento do conteúdo do ponteiro `p`.
- exemplo-ponteiros-02.c

- A operação de subtração funciona de maneira semelhante à operação de soma.
- É possível também realizar operações relacionais entre ponteiros. Por exemplo, é possível verificar se dois ponteiros têm o endereços iguais (==) ou diferentes (!=). Ainda é possível verificar se o endereço de um ponteiro é maior (>) ou menor (<) que o endereço de outro ponteiro. Um ponteiro maior que o outro significa que aquele está à direita, enquanto que um ponteiro menor do que o outro significa que aquele está à esquerda deste.
- Verificação de aprendizado:
 - Explique a diferença entre: `p++; (*p)++; *(p++);`
- Ponteiros para ponteiros
 - Um ponteiro para ponteiro guardar o endereço de um ponteiro, sendo que este guarda o endereço de uma variável. Ou seja, um ponteiro para ponteiro guardar um endereço que aponta para outro endereço.
 - Sintaxe: `<tipo-variável> **<nome-ponteiro>;`
 - Não existe limitação desse recurso, ou seja, é possível que ponteiro aponte para outro e este aponte para outro ponteiro. Poderíamos ter uma quantidade indeterminada de referências de ponteiros.
 - exemplo-ponteiros-05.c
- Cuidado no uso de ponteiros
 - O principal cuidado do programador no uso de ponteiros é saber para onde ele está apontando, ou seja, nunca utilize um ponteiro que não foi inicializado.
 - exemplo-ponteiros-06.c
- Ponteiros ainda são normalmente utilizados para manipular strings e vetores/matrizes. Isto será detalhado mais adiante.

Vetores e Matrizes

- Vetores nada mais são do que matrizes unidimensionais.
- Vetores e matrizes são estruturas de dados homogêneas porque todos os elementos que possuem são do mesmo tipo.
- Sintaxe de vetor: `<tipo-variável> <nome-variável> [tamanho];`
- O compilador reserva um espaço de memória suficiente para armazenar toda a estrutura.
- A numeração começa sempre em zero.
- Strings
 - Strings são vetores de caracteres.
 - O último elemento da string possui o valor `'\0'`.
 - Sintaxe de string: `char <nome-string> [tamanho];`
 - Existem diversas funções em C para manipular strings.
 - Comparação entre duas strings
 - Não é possível comparar duas strings através do sinal de igualdade (==). Para isso, é necessário usar a função `strcmp`.
 - Sintaxe: `strcmp (string1, string2);`
 - A função `strcmp` retorna 0 se as strings forem idênticas e retorna um valor diferente de 0 se não forem idênticas.
 - exemplo-strcmp.c
 - Cópia de uma string
 - A cópia de uma string deve ser feita através da função `strcpy`.
 - Sintaxe: `strcpy (destino, origem);`
 - exemplo-strcpy.c
 - Concatenação entre strings
 - A função `strcat` é utilizada para concatenar duas strings.
 - Sintaxe: `strcat (destino, origem);`
 - A string permanecerá inalterada e será anexada ao final da string de destino.

- exemplo-strcmp.c
- Tamanho de uma string
 - A função strlen retorna o tamanho de uma string (quantidade de caracteres).
 - O terminador não é considerado ('\0').
 - exemplo-strlen.c
- Matrizes
 - A forma de declaração de matrizes bidimensionais é bastante parecido com a declaração de vetores.
 - Sintaxe: <tipo-variável> <nome-variável> [num-linhas] [num-colunas];
 - exemplo-maior-elemento-matriz.c
 - As matrizes em C podem ter mais de duas dimensões (multidimensionais).
 - Sintaxe: <tipo-variável> <nome-variável> [tam1] [tam2] ... [tamN];
 - As matrizes, assim como os vetores, podem ser inicializados diretamente. Exemplos:
 - float vet [3] = {1.2, 3.4, 5.7};
 - int mat [2] [3] = {1, 2, 3, 4, 5, 6};
 - char vet [4] = {'c', 'a', 's', 'a', '\0'};
 - char vet [4] = {"casa"};
 - É possível também inicializar as matrizes, assim como os vetores, sem especificar o tamanho dessas estruturas. Por exemplo:
 - char vet [] = "casa";
 - int mat [] [2] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
- Matrizes (vetores) e ponteiros
 - O nome da variável que é utilizada na declaração é, na verdade, um ponteiro para o tipo da variável da matriz. É como se a variável fosse um ponteiro apontando para o primeiro elemento do vetor.
 - Assim, o vetor pode ser acessado como vet[posicao] ou *(vet + posicao).
 - Isso justifica o fato dos vetores começarem em zero, ou seja, *vet é equivalente a vet[0].
 - Exemplo vantagem no uso de ponteiros para manipulação de matrizes:
 - exemplo-ponteiros-matriz.c
 - É importante destacar que o ponteiro é uma variável, mas o nome de um vetor não é uma variável, não sendo possível realizar operações como: vet = vet + 2 ou vet++;
 - O nome de um vetor é um ponteiro constante.
 - exemplo-ponteiro-vetor.c
 - Nesse exemplo é possível perceber que pt[2] é equivalente a *(p+2).
 - A declaração &vet [indice] é válida e retorna o endereço do elemento na posicao do indice, ou seja, o ponteiro vet tem o endereço &vet[0].
 - É possível também criar vetores de ponteiros. Por exemplo, a declaração int *vetpt [10] é um vetor que armazena 10 ponteiros para o tipo inteiro.

Funções

- Uma função é um bloco de código de programa que pode ser usado diversas vezes em sua execução.
- Forma geral:

```
<tipo-retorno> <nome-função> (<lista-argumentos>) {
    lista de comandos;
}
```

- Argumentos são as entradas que uma função recebe.
- scanf e printf são exemplos de funções que têm argumentos de entrada.
- exemplo-funcao01.c
- Retorno de funções

- A Linguagem C não faz distinção entre funções e procedimentos, pois em C existe apenas funções.
- Se não for especificado o tipo de retorno, o compilador C entende que o tipo de retorno será inteiro.
- `main()` é a função principal de qualquer programa em C e o seu retorno sempre será um valor inteiro.
- O valor de retorno de uma função deve ser compatível com o tipo de retorno declarada para a função.
- O comando `return` indica o valor que a função irá retornar.
- exemplo-funcao-produto.c
- O comando `return` na função `main` indica a finalização do programa.
- É possível ter mais de um `return` em uma mesma função.
- exemplo-varios-returns.c
- Tipo `void`
 - O tipo `void` pode ser utilizado nas funções que não retornam valores ou que não possuem parâmetros de entrada.
 - A função `main` também pode ser do tipo `void`.
- Protótipos de funções
 - Protótipos de funções são declarações de funções, sem que haja uma implementação.
 - A sintaxe do protótipo: `<tipo-retorno> <nome-funcao> (<lista-parâmetros>);`
 - exemplo-prototipo-funcao.c
- Bibliotecas em C
 - As bibliotecas em C são arquivos com a extensão `.h` ou também conhecidos como arquivos de cabeçalho.
 - Os arquivos de cabeçalho são referenciados no início do programa através do comando `include`.
 - Através desse tipo de arquivo, é possível criar funções e colocá-las à disposição do programador.
 - exemplo-uso-bibliotecas.h
- Escopo de variáveis
 - Escopo é o conjunto de regras que determinam o uso e a validade de variáveis nas diversas partes do programa.
 - Variáveis locais
 - Estas variáveis são aquelas que só têm validade dentro do bloco no qual são declaradas.
 - exemplo-variaveis-locais.c
 - Parâmetros formais
 - Estes são declarados como sendo entradas de uma função.
 - O parâmetro formal é uma variável local da função.
 - Esses parâmetros tomam apenas uma cópia dos valores passados para a função.
 - É possível alterar o valor do parâmetro formal, pois esta não terá efeito na variável que foi passada à função.
 - Variáveis globais
 - Variáveis globais são declaradas fora de todas as funções do programa.
 - Elas são conhecidas e podem ser alteradas por todas as funções do programa.
 - Quando uma função tem uma variável local com o mesmo nome de uma variável global a função dará preferência à variável local.
 - exemplo-variavel-local-global.c
 - As variáveis globais devem ser evitadas.
 - Outro tipo de passagem de parâmetros para uma função ocorre quando alterações nos parâmetros formais, dentro da função, alteram os valores dos parâmetros que foram passados para a função. Esse tipo de chamada de função tem o nome de chamada por

- referência.
- A linguagem C só faz chamada por valor.

Estruturas

- As estruturas são coleções de variáveis de vários tipos, que são referidas dentro do programa pelo mesmo nome.
- Sintaxe:

```
struct <nome-estrutura> {  
    <tipo1> <nome1>;  
    <tipo2> <nome2>;  
} <lista-variáveis-estrutura>;
```

- <nome-estrutura> define a estrutura e <lista-variáveis-estrutura> é uma lista separada por vírgulas de variáveis que têm as mesmas características.
- Exemplo:

```
struct estruturaAluno {  
    char nome[20];  
    int matricula;  
    float nota;  
} aluno1, aluno2;
```

- Para se fazer referência a um dos elementos da estrutura utiliza-se o nome de uma das variáveis da <lista-variáveis-estrutura> seguido de um ponto e o nome do elemento. Exemplo:
aluno1.nota = 8.0.
- exemplo-estrutura-aluno.c
- Vetores/Matrizes de estruturas
 - Uma estrutura é qualquer tipo de dado em C. Por isso, é possível criar vetores/matrizes do tipo criado pela estrutura.
 - Exemplo: struct estruturaAluno alunos[10];
- Passagem de estruturas como parâmetro de uma função
 - É possível passar só um elemento da estrutura ou toda a estrutura para a função.
 - No primeiro caso, a utilização é igual a uma variável comum. No segundo caso, deve-se declarar como parâmetro da função apenas o nome da variável da estrutura a ser enviada.
 - Exemplo: void preencherNome (struct tipoAluno aluno);
 - exemplo-estrutura-funcao.c
- É possível também renomear o nome dos tipos pré-definidos e também das estruturas criadas. Exemplo:
 - typedef int Inteiro;
 - typedef struct estruturaAluno tipoAluno;
 - exemplo-typedef.c

Referências

MONTGOMERY, Eduard. *Programando com C – Simples e Prático*. Editora Alta Books: Rio de Janeiro, 2006.

SANTOS, Henrique José. *Curso de Linguagem C*. Universidade Federal de Minas Gerais, 1997.