Programação de Software Básico

Aula 1: Linguagem C – operadores e estruturas

Apresentação

Uma tarefa difícil na área de computação é convencer um estudante que aprender uma nova linguagem de programação, ou usar uma linguagem que não é a preferida dele, é necessário e essencial dentro de uma disciplina. Quando se trata de uma linguagem que para alguns está ultrapassada, como a linguagem C, a tarefa é ainda mais difícil.

Talvez o melhor argumento seja: C não é uma linguagem difícil de aprender, então todos os benefícios de aprendê-la serão bem aproveitados. Isto será visto ou recordado por nós nesta primeira aula, onde estabeleceremos os conceitos básicos da linguagem que irá nos acompanhar durante esta disciplina, permitindo que a programação de softwares básicos seja assimilada.

Você pode utilizar o compilador C de sua preferência para esta primeira aula, mas indicamos o compilador DEV C++ na versão 5.11, que será usado em boa parte da disciplina. É só baixá-lo.

Objetivos

- Identificar os itens básicos da linguagem C;
- Estruturar programas em linguagem C;
- Desenvolver programas em C com estruturas de condição e repetição.

Importância da linguagem C

Existem muitas razões para o aprendizado de C ser fundamental. Muitos a consideram até a mãe de todas as linguagens de programação.

Ela foi projetada para implementar o Sistema Operacional Unix, ficando próxima ao sistema operacional, o que a torna uma linguagem eficiente devido ao seu hábil gerenciamento de recursos no nível do sistema.

Outro ponto importante é que essa linguagem não é limitada, mas amplamente utilizada em:

- Sistemas operacionais.
- Compiladores de linguagem.
- Drivers de rede.
- Interpretadores de linguagem.
- Áreas de desenvolvimento de utilitários de sistema.
- Sistemas embarcados (embutidos).

Veja outras vantagens de C:

1

Onipresente

Qualquer que seja a plataforma, C provavelmente está disponível.

2

Portável

Um programa em C compila com modificações mínimas em outras plataformas – às vezes até funciona de imediato.

3

Simples

C é muito simples de aprender e praticamente não requer dependências. Basta um simples PC com o compilador e tudo está pronto para criar programas.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Estrutura de um programa em C e processo de compilação

C é uma linguagem considerada de nível intermediário e precisa de um compilador para criar um código executável e para que o programa possa funcionar em uma máquina.

Compilação é processo de tradução do código fonte escrito para um código de máquina. É feita por um software especial conhecido como compilador, que verifica o código-fonte em busca de qualquer erro sintático ou estrutural e gera um código-objeto com extensão .obj (no Windows) ou .o (no Linux), se o código-fonte estiver livre de erros.

Toda a compilação é dividida em quatro etapas:

1. Pré-processamento.
2. Compilação.
3. Montagem (assembler).
4. Vinculação (linker).

A figura 1 descreve todo o processo de compilação em C.

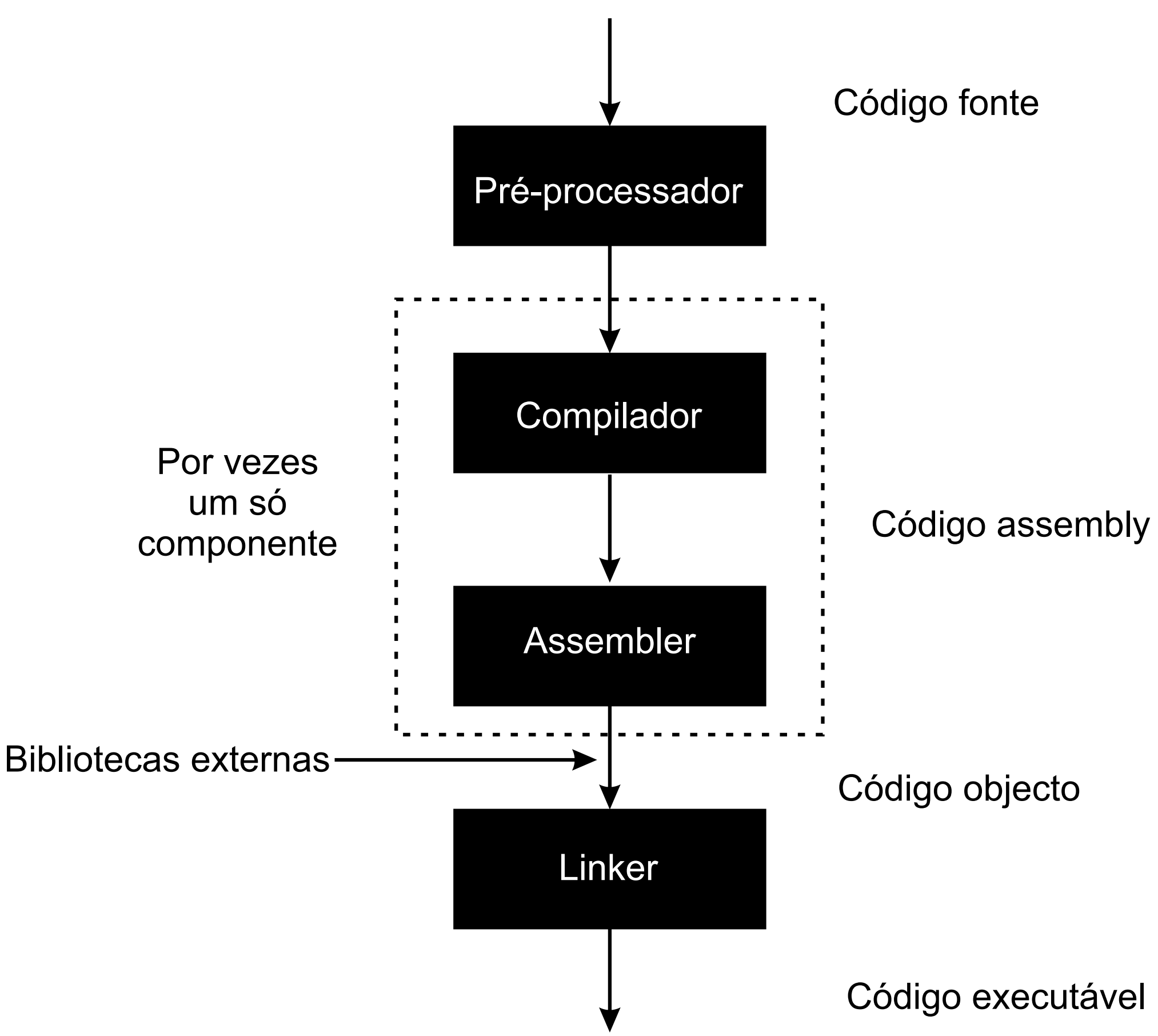


Figura 1: Processo de compilação de um programa

Uma IDE, ou Ambiente de Desenvolvimento Integrado (Integrated Development Environment), reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo, disponibilizando todo o processo de compilação no apertar de um botão.

Exemplo

Podemos detalhar o processo exemplificando a compilação em Linux de um programa em C simples como o abaixo, de nome **compilacao.c**, que escreve na tela a frase “Ola!”:

```
#include <stdio.h>
int main()
{
printf("Ola!");
    return 0;}
```

Para compilar o programa acima abre-se o prompt de comando e pressiona-se o comando abaixo:

```
gcc -save-tempscompilacao.c -o compilacao
```

A opção -save-temps preservará e salvará todos os arquivos temporários criados durante a compilação em C. Ele gerará quatro arquivos no mesmo diretório:

compilacao.i (gerado pelo pré-processador).	compilacao.s (gerado pelo compilador).
compilacao.o (gerado pelo montador).	compilacao (no Linux gerado pelo linker) ou (compilacao.exe no Windows)

Agora, entenda o papel de cada elemento do processo de compila;cão:

Clique nos botões para ver as informações.

Pré-processador

O pré-processador é um pequeno software que aceita o arquivo-fonte C e executa as tarefas abaixo.

- Remove comentários do código-fonte.
- Faz a expansão dos arquivos de cabeçalho incluídos.
- Gera um arquivo temporário com a extensão **.i.**após o pré-processamento.Ele insere o conteúdo dos arquivos de cabeçalho no arquivo de código-fonte. O arquivo gerado pelo pré-processador é maior do que o arquivo de origem original.

Compilador

Na próxima fase da compilação C, o compilador entra em ação. Ele aceita o arquivo pré-processado temporário **nome_do_arquivo.i** gerado pelo pré-processador e executa as seguintes tarefas:

- Verifica o programa C para erros de sintaxe.
- Traduz o arquivo em código intermediário, ou seja, em linguagem assembly.
- Otimiza,opcionalmente, o código traduzido para melhor desempenho.
- Gera um código intermediário na linguagem assembly,após a compilação,como **nome_do_arquivo.s**. É a versão de montagem do código-fonte.

Montador (Assembler)

Passando para a próxima fase de compilação, o assembler aceita o código-fonte compilado (nome_do_arquivo.s) e o traduz em código de máquina de baixo nível. Após a montagem bem-sucedida, gera o arquivo **nome_do_arquivo.o** (no Linux) ou **nome_do_arquivo.obj** (no Windows) conhecido como arquivo objeto. No nosso caso, gera o arquivo **compilacao.o**.

Vinculador (Linker)

Finalmente, o linker entra em ação e executa a tarefa final do processo de compilação. Aceita o arquivo intermediário **nome_do_arquivo.o** gerado pelo assembler.

Ele liga todas as chamadas de função com sua definição original. O que significa que a função printf () é vinculada à sua definição original. O vinculador gera o arquivo executável final.

Variáveis e tipos de dados

Na programação, uma variável é um contêiner (área de armazenamento) para armazenar dados.

Para indicar a área de armazenamento, cada variável deve receber um nome exclusivo (identificador). Os nomes de variáveis são apenas a representação simbólica de um local de memória.

Exemplo

```
int resultado = 95;
```

Aqui, **resultado** é uma variável do tipo inteiro. Para esta variável, é atribuído um valor inteiro, 95.

O valor de uma variável pode ser alterado, como abaixo. Daí o nome, **variável**.

```
char ch = 'a';
// algum código
ch = 'l';
```

Regras para nomear uma variável

Um nome de variável pode ter letras, dígitos e símbolo “_”.A primeira letra de uma variável deve ser uma letra ou o “_”.

Atenção

Não há nenhuma regra sobre o tamanho que um nome de variável (identificador) pode ter. No entanto, podemos ter problemas em alguns compiladores se o nome da variável tiver mais de 31 caracteres.

C é uma linguagem fortemente tipada ou tipificada. Isso significa que o tipo da variável não pode ser alterado depois de declarado.

Exemplo

```
intnumero  = 5;                 // variável inteira
numero = 5.5;                   // erro
floatnumero ;                   // erro
```

Aqui, o tipo de variável numérica é int. Você não pode atribuir um valor de ponto flutuante (5.5) a essa variável. Além disso, você não pode redefinir o tipo da variável para float.

A propósito, para armazenar valores com casas decimais em C, você precisa declarar seu tipo para double ou float.

Constantes

Uma constante é um valor (ou um identificador) cujo valor não pode ser alterado em um programa.

Exemplo

1, 2.5, 'c' etc.

Aqui, 1, 2.5 e 'c' são constantes literais. Não se pode atribuir valores diferentes a esses termos.

constfloat PI = 3,14;

Observe que adicionamos a palavra-chave **const.**

Aqui, PI é uma constante simbólica. Na verdade, é uma variável, no entanto, seu valor não pode ser alterado.

Tipos de constantes

Veja os tipos de constantes que podem ser usadas em C:

- **Constantes inteiras.**
- **Constantes de ponto flutuante.**
- **Constantes de caracteres.**

Uma constante de caractere é criada, colocando-se um único caractere entre aspas simples.

Por exemplo: 'a', 'm', 'F', '2', '}' etc.

- **Sequências de escape**

Às vezes, é necessário usar caracteres que não podem ser digitados ou que tenham significado especial na programação C. Para usar esses caracteres, a sequência de escape é usada.

Por exemplo: \n é usado para nova linha. \t como tabulação horizontal. A barra invertida (\) faz com que se escape do modo normal, em que os caracteres são manipulados pelo compilador.

- **Stringliteral**

Uma string literal é uma sequência de caracteres entre aspas duplas.

Exemplo

```
"legal"          // constante de string
""               // constante de cadeia nula
"      "         // constante de seis espaços em branco
"A"              // constante de string com caractere único
"Resultado eh\n"     // imprime string com nova linha
```

- **Enumerações**

A palavra-chave **enum** é usada para definir tipos de enumeração.

Exemplo

```
enum cor {amarelo, verde, preto, branco};
```

Aqui, a cor é uma variável e amarelo, verde, preto e branco são as constantes de enumeração com valor 0, 1, 2 e 3, respectivamente.

Pode-se definir constantes simbólicas usando-se também a palavra **#define**.

Tipos de dados e modificadores

São 5 os tipos de dados básicos em C:

char	Caractere
int	inteiro
float	real de precisão simples
double	real de precisão dupla
void	vazio (sem valor)

Comentário

Com exceção de void, os outros tipos de dados primitivos podem ter modificadores. Os modificadores alteram o tamanho do tipo de dado ou sua forma de representação.

Os modificadores são:

1

igned

caracere

2

unsignd

ingteiro

3

long

longo

4

short

curto

Atuando nos tipos de dados, os modificadores alteram o alcance destes, como nos exemplos da tabela abaixo:

Palavra-chave	Tipo	Tamanho em bytes	Intervalo
Char	Caractere	1	-128 a 127
signed char	Caractere com sinal	1	-128 a 127
unsigned char	Caractere sem sinal	1	0 a 255
Int	Inteiro	2	-32.768 a 32.767
signedint	Inteiro com sinal	2	-32.768 a 32.767
unsignedint	Inteiro sem sinal	2	0 a 65.535
short int	Inteiro curto	2	-32.768 a 32 767
signed short int	Inteiro curto com sinal	2	-32.768 a 32.767
unsigned short int	Inteiro curto sem sinal	2	0 a 65.535
longint	Inteiro long	4	-2.147.483.648 a 2.147.483.647
signedlongint	Inteiro longo com sinal	4	-2.147.483.648 a 2.147.483.647
unsignedlongint	Inteiro longo sem sinal	4	0 a 4.294.967.295
float	Ponto flutuante com precisão simples	4	3.4 E-38 a 3.4E+38
double	Ponto flutuante com precisão simples	8	1.7 E-308 a 1.7E+308
longdouble	Ponto flutuante com precisão dupla longo	16	3.4E-4932 a 1.1E+4932

Especificadores de tipo de classe de armazenamento

Os especificadores de classe de armazenamento são usados para informar ao compilador como a variável deve ser armazenada.

Esses especificadores são:

Clique nos botões para ver as informações.

extern

Diz ao compilador que as variáveis que seguem foram declaradas em outro lugar, evitando que a mesma variável seja armazenada várias vezes.

Assim, todas as variáveis globais podem ser declaradas em um único arquivo, usando-se declarações **extern** nos outros.

static

Ao contrário das variáveis globais, estas não são reconhecidas fora de sua função ou arquivo, mas seus valores são mantidos entre as chamadas.

register

O especificador informa ao compilador que a variável deve ser armazenada em um registrador da CPU, e não na memória. Isso aumenta consideravelmente a velocidade de acesso.

Como alguns tipos de dados não cabem em um registrador, o compilador trata variáveis register de forma que o acesso seja o mais rápido possível.

Somente variáveis locais e parâmetros formais podem ser register.Variáveis globais não são permitidas.

Operadores em linguagem C

Operadores aritméticos nativos da linguagem C são a base para os cálculos mais complexos. Em geral, operadores matemáticos que não estão entre os operadores nativos da linguagem, como a raiz quadrada ou a potência, são disponibilizados por meio de funções programadas em C.

Os operadores são mostrados na tabela abaixo:

Operador	Operação matemática
+	Adição
–	Subtração
*	Multiplicação
/	Divisão
%	Módulo (obtém o resto da divisão)
––	Decremento unário
++	Incremento unário

Operadores de atribuição

Para armazenar dados em variáveis, é preciso fazer atribuições. Para isso, deve-se utilizar o principal operador de atribuição, que, na linguagem C, é o sinal de igualdade, "=".

Veja outros operadores de atribuição na tabela:

Operador	Operação matemática
+ =	Atribuição acumulando por soma
– =	Atribuição acumulando por subtração
* =	Atribuição acumulando por multiplicação
/ =	Atribuição acumulando por divisão
% =	Atribuição acumulando por módulo

Entrada e saída

Para um programa de computador, é fundamental a interação com dispositivos de entrada e de saída.

A configuração mais comum é o teclado como dispositivo de entrada e o monitor como dispositivo de saída.

Função printf

Uma função é um conjunto de comandos agrupados em um bloco, que recebe um nome e através deste pode ser chamado, permitindo o reaproveitamento de código já construído.

A função printf() permite realizar a impressão de textos no monitor, ou seja, é responsável pela saída de informações, utilizando a sintaxe: printf("formato", argumentos);

O primeiro argumento da função é obrigatório, ou seja, no mínimo deve-se informar um texto para ser impresso.

Os próximos argumentos são opcionais, pois nem sempre é necessário apresentar uma informação em conjunto do texto.

O uso da função com mais de um argumento requer também o uso de formatadores de tipo, conforme a tabela abaixo.

Formato	Tipo da variável
%c	Caracteres
%d	Inteiros
%e	Ponto flutuante, notação científica
%f	Ponto flutuante, notação decimal
%lf	Ponto flutuante, notação decimal double
%g	O mais curto de %e ou %f
%o	Saída em octal
%s	String char
%u	Inteiro sem sinal
%x	Saída em hexadecimal (0 a f)
%X	Saída em hexadecimal (0 a F)
%ld	Saída em decimal longo

Exemplo

```
float total = 4 + 5.7;
printf("Total: %f", total);
```

Função scanf

Similar à função printf(), a função scanf() também suporta uma quantidade "n"de argumentos e permite que os dados digitados pelo usuário sejam armazenados nas variáveis do programa.

Exemplo

```
intmat;
scanf("%d", &mat);
```

Comentário

Na linha 2, foi realizado o acionamento da função scanf(). No primeiro argumento foi informado o formato entre aspas, "%d", conforme apresentado para o tipo de dado inteiro. E o segundo argumento é o caractere ‘&’ acompanhado do nome da variável.

A partir do segundo argumento, deve-se informar o endereço de memória no qual o dado será armazenado.Por isso, foi utilizado o caractere & acompanhado do nome da variável.

Estruturas condicionais

Grande parte das vezes, os problemas dão origem a programas que necessitam possuir variados fluxos ou caminhos. Para que as instruções em um programa tomem um caminho diferente, é necessário que haja uma instrução responsável pela decisão de qual caminho tomar.

Cláusula switch

O switch é um comando com possibilidades mais simplificadas que o if-else.Desta forma, permite apenas a comparação de igualdade com variáveis do tipo int, char e long.

O switch é vantajoso quando é necessário fazer muitas comparações, pois oferecerá maior agilidade na implementação.

Veja a sintaxe:

```
switch (expressao) {
case constante1:
instrucoes1;
break;
case constante2:
instrucoes2;
break;
...
default:
instrucoes;
}
```

Para utilizar o switch, basta substituir a palavra expressao pelo nome da variável que terá sua expressão avaliada. Substituir a palavra constante1 pela constante a ser comparada com o conteúdo da variável em expressao, e instrucoes1 pelas instruções que se pretende executar caso a comparação seja verdadeira.

A cláusula **default** funciona como o último **else** em um conjunto de instruções if, ou seja, se nenhuma condição anterior é verdadeira, então as instruções em default serão executadas. A cláusula default é opcional.

A cláusula **break** é responsável pela parada na execução das instruções, pois, caso não seja colocada, as validações presentes no switch continuarão a ser executadas, mesmo que um case já tenha resultado em verdadeiro.

Estruturas de repetição ou iteração

Estruturas de repetição permitem que um conjunto de instruções seja repetido até que se faça a condição desejada.

Cláusula for

A cláusula for é muito útil quando se deseja repetir uma ou várias instruções por um número n de vezes.

Embora o for possibilite variações, o formato de uso mais comum é utilizar uma variável que é incrementada e verificada a cada iteração.Assim, quando a variável atinge um determinado valor, o laço se encerra.

Veja a sintaxe:

```
for (inicializacao; condicao de laco ou parada; incremento) {
instrucao01;
instrucao02;
...
instrucaoN;
}
```

Comentário

Neste caso, foi delimitado o início e o término do bloco de instruções pelas chaves, e, entre as chaves, estão as n instruções que serão executadas no laço.

A estrutura de iteração for permite também o uso de laços aninhados (encaixados) que são úteis quando são necessárias iterações dentro de outras.

Cláusula while

Diferente do for, o while geralmente é empregado quando não se pode determinar com certeza quantas vezes um bloco de comandos será executado.

A condição do while é definida de forma muito similar à definição da condição no if. A diferença é que no if o objetivo é desviar o caminho de execução para um fluxo de instruções ou outro; no while o objetivo será manter a execução de um bloco de instruções em execução, assim como no for.

Veja a sintaxe:

```
while (condicao de laco ou parada) {
instrucao01;
instrucao02;
...
instrucaoN;
}
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Introdução às estruturas de dados

Em todos os programas são manipulados dados. Por isso, é conveniente que esses dados sejam armazenados de forma que sua utilização se torne mais fácil e eficiente. É daí que surge o estudo das Estruturas de Dados.

O segredo de muitos programas rápidos está na maneira como seus dados são organizados durante o processamento, ou seja, na estrutura de dados empregada pelo programa.

Fonte: Shutterstock.

Vetores ou arrays

Vetor é um tipo de estrutura de dados que pode armazenar uma coleção sequencial e de tamanho fixo de elementos do mesmo tipo.

Um vetor é usado para armazenar uma coleção de dados, mas geralmente é mais útil pensar em um vetor como uma coleção de variáveis do mesmo tipo.

Exemplo

Exemplo de declaração

intnumero[100];

Em vez de declarar variáveis individuais, como numero0, numero1, ... e numero99, você declara uma variável tipo vetor e usa numero[0], numero[1] e ..., numero[99] para representar variáveis individuais. Um elemento específico em um vetor é acessado por um índice.

Todas os vetores consistem em locais de memória contíguos. O endereço mais baixo corresponde ao primeiro elemento e o endereço mais alto ao último elemento.

Exemplo

Programa em C que armazena 10 números inteiros fornecidos pelo usuário em um vetor NUM e imprime uma lista dos números lidos.

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

void main() {
int i, NUM[15];

//lendo os valores
for (i=0; i<15; i++) {
printf("Informe um numero: \n");
scanf("%d", &NUM[i]);
}
//imprimindo os valores
for (i=0; i<15; i++) {
printf("Numero: %d \n", NUM[i]);
}
}
```

Matrizes

Uma matriz, ou vetor bidimensional é, em essência, uma lista de vetores unidimensionais. Para declarar uma matriz de tamanho x linhas e y colunas, pode-se escrever algo da seguinte maneira

```
tipo matrizNome [x][y];
```

Onde **tipo** pode ser qualquer tipo de dado válido em C e matrizNome será um identificador C válido.

Ponteiros

Os ponteiros são recursos poderosos de programação C e (C ++) que a diferencia de outras linguagens de programação populares, como Java e Python.

Ponteiros são usados no programa em C para acessar a memória e manipular o endereço.

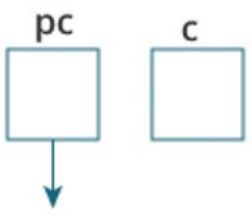
Em C, pode-se criar uma variável especial que armazena o endereço (em vez do valor). É essa variável que é chamada de **variável de ponteiro** ou simplesmente um ponteiro. Para declarar um ponteiro:

```
TipodeDado
*nome_variavel_ponteiro;
int *p;
```

Comentário

A declaração acima define p como variável ponteiro do tipo int.

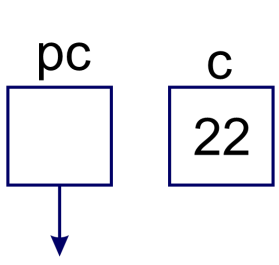
Operador (&) e operador (*)

O operador & é chamado de **operador de referência**. Ele fornece o endereço de uma variável. Da mesma forma, existe outro operador que obtém o valor do 

endereço, que é chamado de operador de remoção de referência *[1] .

Exemplo

int *pc, c;



Aqui, um ponteiro pc e uma variável normal c, ambos do tipo int, são criados.

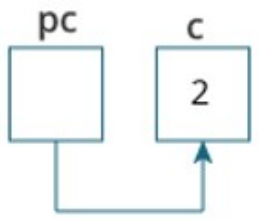
Desde que pc e c não são inicializados, o ponteiro pc aponta para nenhum endereço ou um endereço aleatório. E a variável c tem um endereço, mas contém um valor aleatório de lixo.

```
c = 22;
```

Comentário

Isso atribui 22 à variável c, isto é, 22 é armazenado na localização de memória da variável c.

```
pc = &c;
```

Isso atribui o endereço da variável c ao ponteiro pc. O valor de pc é o mesmo que o 

endereço de c, e o conteúdo do pc é 22 também.

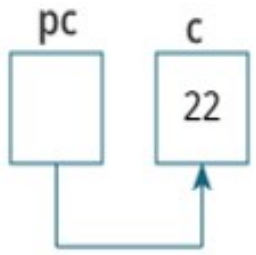
```
*pc = 2;
```

Isso altera o valor na localização da memória apontada pelo ponteiro pc para 2. Como o endereço do ponteiro pc é o mesmo que o endereço de c, o valor de c também é alterado para 2.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Alocação dinâmica de memória

Um vetor é uma coleção de números fixos de valores de um único tipo. Ou seja, você precisa declarar o tamanho de um vetor antes de poder usá-lo. Às vezes, o tamanho do vetor pode ser insuficiente.

Para resolver esse problema, pode-se alocar memória manualmente durante o tempo de execução. Isso é conhecido **como alocação dinâmica de memória** na 

programação C.

Sintaxe de malloc()

O nome "malloc" significa alocação de memória.

A função malloc() reserva um bloco de memória do número especificado de bytes. E, ele retorna um ponteiro do tipo void que pode ser convertido em ponteiro de qualquer forma.

```
ptr = (tipo *) malloc
(tamanho do byte)
```

Exemplo

ptr = (int *) malloc (100 * sizeof (int));

- **free()**

Sintaxe de malloc()

free(ptr);

Esta declaração libera o espaço alocado na memória apontada por ptr.

Atividade

1. Vários elementos de um programa em C, como variáveis, funções e matrizes, precisam ser identificados por nomes, chamados de forma geral por **Identificadores**. Algumas palavras reservadas, ou palavras-chave, são usadas para definir ou modificar como os elementos identificados serão tratados pelo compilador.

Indique, nas frases abaixo, qual das palavras-chave, numeradas de 1 a 5, estabelece a função correspondente na linguagem C.

const 1 static 2 extern 3 register 4 void 5

a) Usada para estabelecer que a função identificada logo após esta palavra-chave não retornará valor.

b) Usada para identificar as variáveis seguintes a esta palavra-chave como já declaradas em outra parte do código.

c) Usada para informar ao compilador que a variável que a segue é permanente, ou seja, seu valor é mantido igual entre cada chamada.

d) Usada para indicar que o valor da variável que se segue pode ser inicializado, mas não pode ser alterado, mesmo por outro comando no código.

e) Usada para acelerar operações por meio da armazenagem do valor da variável em um registrador da CPU.

2. Se declararmos um vetor como **int vet[30]**, a instrução abaixo acessa corretamente os elementos deste vetor?

```
for (j=0; j <= 30; j++)
  vet[j] = j*j;
```

3. Escreva um programa em C que leia um valor inicial A e imprima a sequência de valores do cálculo de fatorial de A (A!) e o seu resultado. Ex: 5! = 5 X 4 X 3 X 2 X 1 = 120.

4. Desenvolva um programa em C que leia a altura de pessoas, cujo número de pessoas é dado pelo usuário. Este programa deverá verificar e mostrar:

a. A menor altura do grupo;

b. A maior altura do grupo.

5. Qual o resultado do programa a seguir, que lida com ponteiros e vetores em C?

```
#include<stdio.h>
int main()
{
   int arr[] = { 2, 3, 4, 5 };
   int* p = (arr + 2);
   printf ("%d", *arr + 15);
   return 0;
}
```

Notas

Operador de remoção de referência[1]

O sinal *, ao declarar um ponteiro, não é um operador de referência. É apenas uma notação semelhante que cria um ponteiro.

Unicórnio[2]

Termo introduzido em 2013 por Aileen Lee, referindo-se a empresas que conseguem algo que parece impossível (como achar um unicórnio), que é ser avaliada em 1 bilhão de dólares antes mesmo de abrir seu capital nas bolsas de valores. E o que elas fazem para conseguir esse feito? Introduzem grandes inovações no mercado, enxergam a riqueza de dados que a internet fornece e conseguem ter o controle deles extraindo valor das transações de dados.

Cibridismo[3]

É estar on e off o tempo todo.

Somos seres ciber-hídridos, ou seja, temos uma constituição biológica, expandida por todas as interfaces tecnológicas que adquirimos, e, cada vez mais, estaremos replicados em todas essas plataformas. Nossos conteúdos, dados pessoais, fotos, vídeos, leituras, tudo o que faz parte da nossa vida está integrado nas interfaces que utilizamos, e não vivemos sem eles. Isso é ser cíbrido.

Referências

AGUIAR, Marcelo O.; FREITAS, Rodrigo. **Introdução ao C em 10 aulas**. 1. ed. Alegre: Marcelo Otone Aguiar, 2016. Disponível em: //repositorio.ufes.br/bitstream/10/6800/1/Introdução_C_10_Aulas.pdf. Acesso em: 21 nov. 2019.

Próxima aula

- Estudo de bibliotecas em C com compiladores diversos;
- Acesso a data e hora;
- Criação de gráficos.

Explore mais

O aprendizado de uma linguagem de programação exige a resolução constante de problemas para exercitar o uso da sintaxe e da lógica de forma adequada. É fundamental que você busque resolver problemas mais comuns para complementar o aprendizado desta aula, principalmente na criação de funções, algo que será muito utilizado durante o curso.

Os materiais abaixo são úteis para esse fim, não deixe de estudá-los:

- Exercícios complementares – Funções.
- Computação Científica em Linguagem C - Um Livro Colaborativo.
- Linguagem C: funções.