

Lógica de Programação: notas de aula

Prof. Jonatha Costa

2024

Organização

① Linguagem C - Introdução

Linguagens e conceitos basilares de programação

Linguagem C

Ambientes de Desenvolvimento Integrado

Funções iniciais

② Estruturas de Controle - C

③ Vetores e Matrizes - C

Objetivo da aula

- Estudar a sintaxe da linguagem C através da criação de cabeçalho, declaração de comentários e variáveis e tipos de dados.

Organização

1 Linguagem C - Introdução

Linguagens e conceitos basilares de programação

Linguagem C

Ambientes de Desenvolvimento Integrado

Funções iniciais

2 Estruturas de Controle - C

O comando if

O comando switch

O comando while

O comando do-while

O comando for

3 Vetores e Matrizes - C

Linguagens de Programação

- **Linguagem de Programação:** Um conjunto de instruções que permitem a comunicação entre o programador e o computador, permitindo criar rotinas e aplicações.
- Existem diferentes tipos de linguagens de programação, com níveis variados de abstração:
 - **Linguagens de Baixo Nível:**
 - Mais próximas do *hardware*, oferecendo controle direto sobre os componentes físicos do computador.
 - São difíceis de ler e escrever para humanos, mas muito eficientes para o desempenho da máquina.
 - Exemplos incluem **Assembly** e **linguagem de máquina** (código binário).
 - Utilizadas em sistemas embarcados, desenvolvimento de drivers e otimizações de *hardware*.
 - **Linguagens de Alto Nível:**
 - Mais próximas da linguagem humana, com uma sintaxe mais fácil de ler e escrever.
 - Abstraem muitos detalhes do *hardware*, permitindo que o programador foque mais na lógica do problema.
 - Exemplos incluem **C**, **Python**, **Java**, e **JavaScript**.
 - São amplamente utilizadas para desenvolvimento de aplicações, websites, *softwares* corporativos e *scripts* de automação.

Conceitos Básicos

- **Variáveis:** Armazenam valores que podem mudar durante a execução de um programa.
- **Funções:** Blocos de código que executam tarefas específicas e podem ser reutilizados.
- **Condicionais:** Permitem tomar decisões baseadas em condições (`if`, `else`).
- **Laços de Repetição:** Executam um bloco de código várias vezes (`for`, `while`).
- **Compilação/Interpretação:** O código pode ser compilado (C) ou interpretado (Python).

Organização

① Linguagem C - Introdução

Linguagens e conceitos basilares de programação

Linguagem C

Ambientes de Desenvolvimento Integrado

Funções iniciais

② Estruturas de Controle - C

O comando if

O comando switch

O comando while

O comando do-while

O comando for

③ Vetores e Matrizes - C

Linguagem C

Instruções basilares

- 1 Toda função em C deve ser iniciada por um abre-chaves '{' e encerrada por um fecha-chaves '}';
- 2 O nome, os parênteses e as chaves são os únicos elementos obrigatórios de uma função (main);
- 3 Todas as instruções devem estar dentro das chaves;
- 4 As instruções em C são sempre encerradas por um ponto-e-vírgula;
- 5 As chaves são também utilizadas para separar um bloco de instruções.

Sintaxe da Linguagem C

Exemplo

```
main( )           // primeira função a ser executada
{                 // início do corpo da função

<comandos>;

}                 // término do corpo da função
```

Sintaxe da Linguagem C: inserindo um comentário

Duas maneiras:

- Uma linha: “ // ”
- Em bloco: “ /* ” e “ */ ”

Comentários

```
/*
```

Lógica de Programação

Este é um comentário em bloco

```
*/
```

```
int contador; // Este é um comentário de uma linha
```

Sintaxe da Linguagem C: regras

- Um identificador não pode ter símbolos gráficos, com exceção do underline (“ _ ”);
- Não pode haver acentuação gráfica (acento grave, acento agudo, til, circunflexo ou cedilha);
- Um identificador não pode ser iniciado por número;
- Não pode ser uma palavra reservada;

Sintaxe da Linguagem C: exemplos

Tabela: Estruturas de sintaxe

| Correto | Incorreto |
|--------------|---------------|
| contator | lcontator |
| Teste23 | oi!gente |
| Alto_Paraiso | Alto..Paraíso |
| __sizeint | _size-int |
| nota1 | lnota |

Fonte: Autor,(2020)

Sintaxe da Linguagem C: expressões dedicadas da linguagem C

Tabela: Expressões dedicadas do C

| | | | |
|----------|--------|--------|----------|
| default | return | else | continue |
| float | switch | if | long |
| register | while | signed | static |
| struct | case | const | void |
| volatile | double | enum | unsigned |
| break | goto | int | char |
| do | short | sizeof | for |

Autor,(2020)

Sintaxe da Linguagem C

Letras maiúsculas e minúsculas são tratadas diferentes.

Exemplos:

- `int variavel; int Variavel; int VaRiAVel;`
- `int VARIABEL;`
- `int variavel, Variavel, VaRiAVel, VARIABEL;`

Tipos Básicos de Dados

O tipo de uma variável define os valores que ela pode assumir e as operações que podem ser realizadas com ela.

Descreve a natureza da informação.

Exemplo:

- Variáveis do tipo int recebem apenas valores inteiros
- Variáveis do tipo float armazenam apenas valores reais

Tipos Básicos de Dados

Figura: Tipo de dados

| TIPO | TAMANHO EM BITS | Início | Fim |
|--------------|-----------------|------------------------|-----------------------|
| char | 8 | 0 | 255 |
| int | 16 | -32.768 | 32.767 |
| unsigned int | 16 | 0 | 65.535 |
| short int | 16 | -32.768 | 32.767 |
| float | 32 | $3,4 \times 10^{-38}$ | $3,4 \times 10^{38}$ |
| double | 64 | $1,7 \times 10^{-308}$ | $1,7 \times 10^{308}$ |
| void | 0 | sem valor | sem valor |

Fonte: DIAS,(2019)

Tipos Básicos de Dados

Um modificador é usado para alterar o significado de um tipo básico para adaptá-lo mais precisamente às necessidades de diversas situações.

Figura: Tipo de dados

| TIPO | TAMANHO EM BITS |
|-----------------------|---|
| <code>signed</code> | Representa tanto números positivos quanto negativos |
| <code>unsigned</code> | Representa números sem sinal |
| <code>short</code> | Determina que a variável tenha um tamanho menor que o tipo modificado |
| <code>long</code> | Determina que a variável tenha um tamanho maior que o tipo modificado |

Fonte: DIAS,(2019)

Declaração de variáveis

- Todas as variáveis em C devem ser declaradas, antes de serem usadas;
- Uma declaração de variável em C consiste no nome de um tipo, seguido do nome da variável, seguido de ponto e vírgula.

Declaração

Ex: tipo_da_variavel lista_de_variaveis;

```
int num;
```

Declaração de variáveis

Podemos inicializar variáveis no momento de sua declaração.

```
tipo_da_variável nome_da_variável = valor;
```

Declaração

```
main( )  
{  
  
    int contador = 0;                //contador de aplicação geral  
    int temp_1, temp_2 = 10;         //variáveis auxiliares  
  
    ...  
  
}
```

Declaração de variáveis

Podemos inicializar variáveis no momento de sua declaração.

Declaração

```
main( )  
{  
  
float nota_prova_a = 8.0;  
float nota_prova_b = 6.0;  
float nota_laboratorio = 10.0;  
float media;  
  
...  
  
}
```

Declaração de variáveis

Podemos inicializar variáveis no momento de sua declaração.

Exemplo

```
main( )  
{  
  
char c; char caract = 'a';  
int nota = 6;  
int var_2;  
float f;  
float media;  
  
}
```

Organização

1 Linguagem C - Introdução

Linguagens e conceitos basilares de programação

Linguagem C

Ambientes de Desenvolvimento Integrado

Funções iniciais

2 Estruturas de Controle - C

O comando if

O comando switch

O comando while

O comando do-while

O comando for

3 Vetores e Matrizes - C

O que é uma IDE?

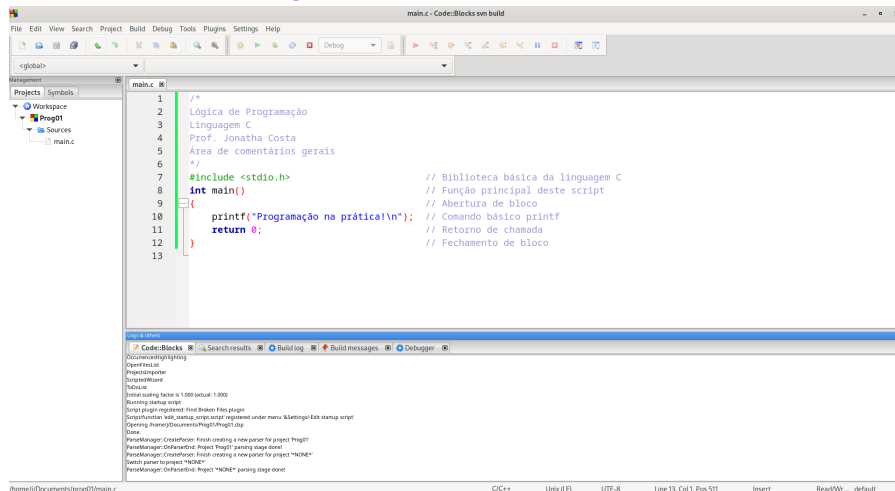
- IDE (*Integrated Development Environment*) é um ambiente de desenvolvimento integrado.
- Facilita o processo de desenvolvimento, oferecendo um conjunto de ferramentas como:
 - Editor de código
 - Compilador
 - Depurador (debugger)
 - Ferramentas de gerenciamento de projeto
- IDEs são projetadas para aumentar a produtividade do programador, integrando funcionalidades em uma interface única.

IDEs adequadas Programação em C

- **Code::Blocks**
 - Interface amigável
 - Suporte a múltiplos compiladores
 - Extensível por meio de plugins
- **Eclipse CDT**
 - Ambiente robusto e poderoso
 - Suporte a projetos de grande escala
 - Extensões para diversas linguagens
- **CLion**
 - Ferramentas avançadas de refatoração e análise de código
 - Excelente suporte a CMake
 - Integração com VCS (Controle de Versão)
- **Visual Studio Code**
 - Leve, com suporte a extensões para C/C++
 - Integração com depuradores e Git
- **Dev-C++**
 - IDE clássica, leve e simples
 - Ideal para iniciantes no aprendizado de C

Exemplo de IDE

Figura: IDE do Code: Blocks IDE



Fonte: AUTOR (2024)

Programação no Code: Bocks IDE

Figura: Destaque de um primeiro código em C via Code: Bocks IDE

The screenshot displays the Code:Blocks IDE interface. The top pane shows a C program in a file named 'cb.c'. The code includes a multi-line comment in Portuguese, followed by an include statement for 'stdio.h', and a 'main' function that prints 'Programação na prática!' and returns 0. The bottom pane shows the execution output, which matches the print statement in the code.

```

1  /******
2  Lógica de Programação
3  Linguagem C
4  Prof. Jonatha Costa
5  Área de comentários gerais
6  *****/
7  #include <stdio.h>           // Biblioteca básica da linguagem C
8  int main()                  // Função principal deste script
9  {                             // Abertura de bloco
10     printf("Programação na prática!\n"); // Comando básico printf
11     return 0;                // Retorno de chamada
12 }                             // Fechamento de bloco
13
14

```

Logs & others

Code::Blocks Search results Build log Build messages Debugger

/home/jj/Documents/cb

```

Programação na prática!
Process returned 0 (0x0)   execution time : 0.006 s
Press ENTER to continue.

```

Fonte: AUTOR (2024)

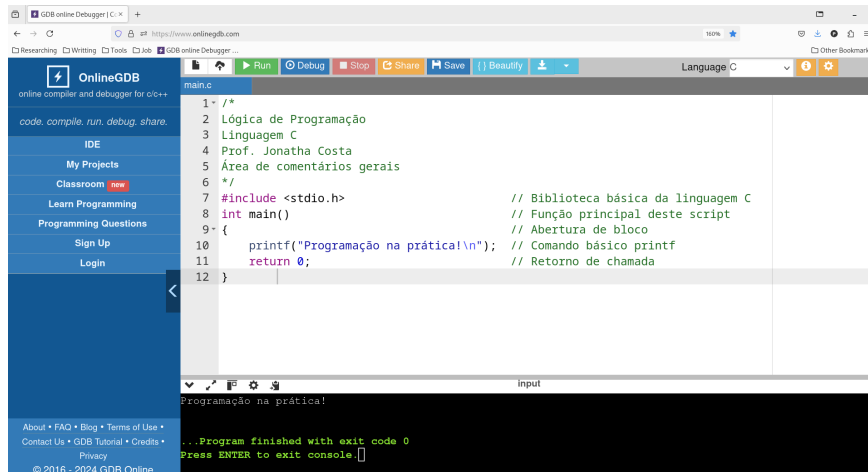
Outras IDEs *on-line*

IDEs *on-line*

- **OnlineGDB** - <https://www.onlinegdb.com/>
 - Depurador e compilador online para C
 - Interface simples e funcional
- **Replit** - <https://replit.com/>
 - IDE online colaborativa
 - Suporte a múltiplas linguagens, incluindo C
- **JDoodle** - <https://www.jdoodle.com/c-online-compiler>
 - Compilador online rápido e leve
 - Suporte a diversas linguagens de programação, incluindo C

IDE do GDB *on-line*

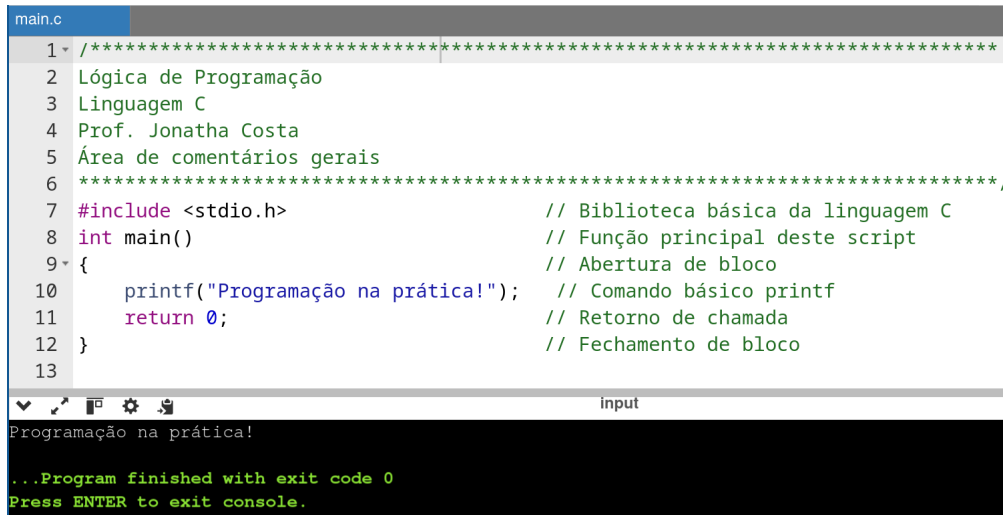
Figura: IDE do GDB *on-line*



Fonte: AUTOR (2024)

Programação no GDB *on-line*

Figura: Destaque do primeiro código em C via CDB online



The screenshot displays an online CDB (GDB) interface. At the top, a tab labeled 'main.c' is active. The main area shows a C program with line numbers 1 through 13. Line 1 is a multi-line comment, and line 6 is another multi-line comment. Lines 7 through 13 contain the program's logic, including a header inclusion and a main function. The program prints 'Programação na prática!' and then returns 0. Below the code editor, there is a toolbar with icons for file operations, settings, and a search icon. The output window at the bottom shows the program's execution, displaying 'Programação na prática!' and then '...Program finished with exit code 0'. The output window also has a toolbar with icons for file operations, settings, and a search icon.

```
1  /*****  
2  Lógica de Programação  
3  Linguagem C  
4  Prof. Jonatha Costa  
5  Área de comentários gerais  
6  *****/  
7  #include <stdio.h>           // Biblioteca básica da linguagem C  
8  int main()                   // Função principal deste script  
9  {                             // Abertura de bloco  
10     printf("Programação na prática!"); // Comando básico printf  
11     return 0;                 // Retorno de chamada  
12 }                             // Fechamento de bloco  
13
```

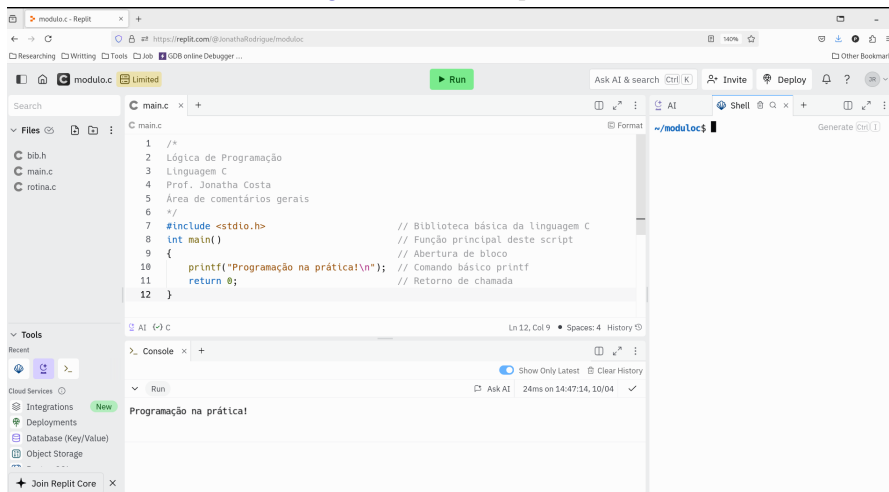
Programação na prática!

...Program finished with exit code 0
Press ENTER to exit console.

Fonte: AUTOR (2024)

IDE do Replit.com

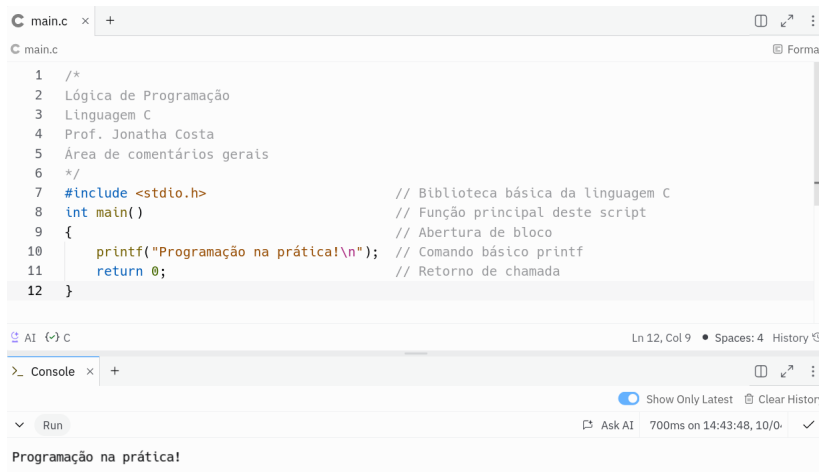
Figura: IDE do Replit.com



Fonte: AUTOR (2024)

Programação no Replit.com

Figura: Destaque do primeiro código em C via Replit.com



The image shows a Replit.com interface with a C code editor. The code is as follows:

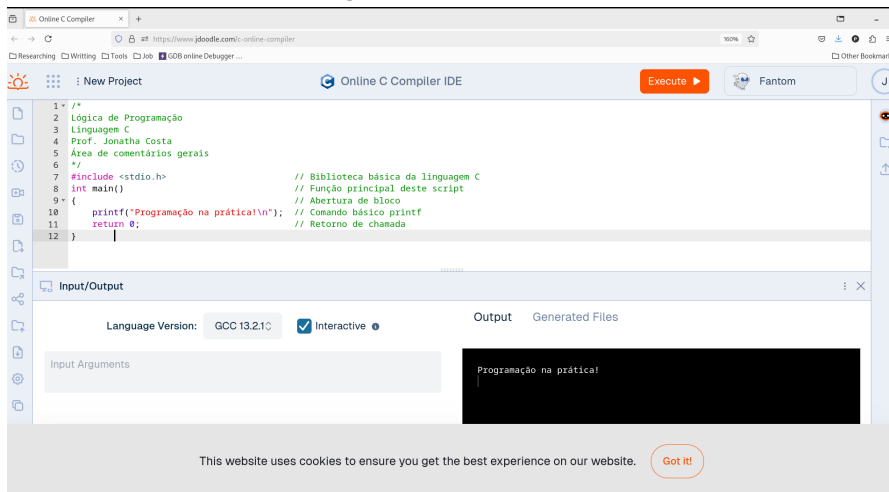
```
1  /*
2  Lógica de Programação
3  Linguagem C
4  Prof. Jonatha Costa
5  Área de comentários gerais
6  */
7  #include <stdio.h>           // Biblioteca básica da linguagem C
8  int main()                  // Função principal deste script
9  {                            // Abertura de bloco
10     printf("Programação na prática!\n"); // Comando básico printf
11     return 0;                // Retorno de chamada
12 }
```

Below the editor, the console output shows "Programação na prática!". The interface includes a file explorer at the top, a console at the bottom, and a status bar at the very bottom.

Fonte: AUTOR (2024)

IDE do Jdoodle

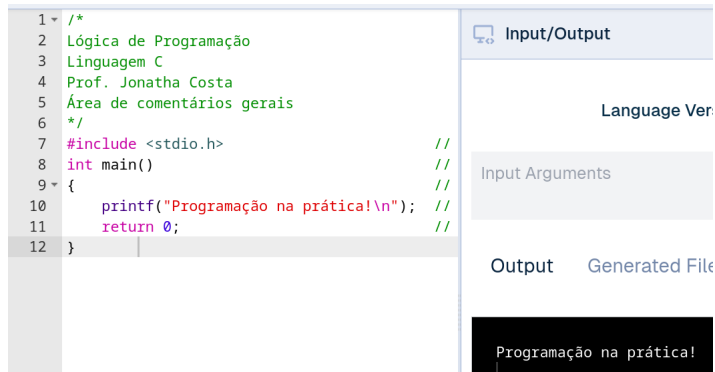
Figura: IDE do Jdoodle



Fonte: AUTOR (2024)

Programação no Jdoodle

Figura: Destaque do primeiro código em C via Jdoodle



The screenshot displays the Jdoodle IDE interface. On the left, a code editor shows a C program with line numbers 1 through 12. The code includes a multi-line comment, a header inclusion, and a main function that prints a message and returns 0. The first line of the comment is highlighted. On the right, a sidebar contains a tab labeled 'Input/Output'. Below this tab, the 'Language Ver:' is set to 'C'. The 'Input Arguments' field is empty. The 'Output' section shows the text 'Programação na prática!' which is highlighted in black. The 'Generated File' section is also empty.

```
1  /*  
2  Lógica de Programação  
3  Linguagem C  
4  Prof. Jonatha Costa  
5  Área de comentários gerais  
6  */  
7  #include <stdio.h> //  
8  int main() //  
9  { //  
10     printf("Programação na prática!\n"); //  
11     return 0; //  
12 }
```

Input/Output

Language Ver: C

Input Arguments

Output Generated File

Programação na prática!

Fonte: AUTOR (2024)

Organização

1 Linguagem C - Introdução

Linguagens e conceitos basilares de programação

Linguagem C

Ambientes de Desenvolvimento Integrado

Funções iniciais

2 Estruturas de Controle - C

O comando if

O comando switch

O comando while

O comando do-while

O comando for

3 Vetores e Matrizes - C

Entrada e Saída de Dados

Função **printf**

Usada para saída de dados, informações e comunicação no monitor do computador

- Necessita da biblioteca `stdio.h`.
Colocar `#include<stdio.h>` no cabeçalho do script
- A mensagem deve estar dentro de aspas duplas “ ”
Exemplo : `printf (“mensagem”);`
- Quando se deseja pular uma linha deve-se colocar “\ **n**”
Exemplo: `printf(“mensagem \b”);`

Entrada e Saída de Dados

Função **printf**: forma geral

```
printf(string_de_controle, lista_de_argumentos);
```

- Teremos, na string de controle, uma descrição de tudo que a função vai colocar na tela.
- Isto é feito usando-se os códigos de controle, veja alguns exemplos:

Tabela: Indexação código - conteúdo

| Código | Significado |
|--------|---------------------|
| %d | inteiro |
| %f | float |
| %c | caractere |
| %s | string |
| %% | coloca na tela um % |

Fonte: Autor,(2020)

Entrada e Saída de Dados

Exemplo com variável char

```
#include <stdio.h>
/*Exemplo da variável char*/
main()
{
    char Ch;
    Ch = 'D';
    printf("%c \n", Ch);
}
```

Entrada e Saída de Dados

Exemplo com variável int

```
#include <stdio.h>
/*Exemplo da variável int*/
main()
{
int num;
num= 10;
printf("O valor é %d \n", num);
}
```

Entrada e Saída de Dados

Exemplo com variável float

```
#include <stdio.h>
/*Exemplo da variável float*/
main()
{
float a = 2.2, b = 3;
printf("a = %f\n b = %f \n", a, b);

}
```

Entrada e Saída de Dados

Função **scanf**

Usada para entrada de dados no computador.

- Necessita da biblioteca `stdio.h`.
Colocar `#include<stdio.h>` no cabeçalho do script
- A mensagem deve estar dentro de aspas duplas “ ”
Exemplo : `scanf(“tipos de dados de entrada”,& variaveis);`
- Exemplo:
Exemplo : `int a; scanf(“%d”, &a);`
Resultado:
Ao digitar um numero no teclado e pressionar enter, a variável `a` assume o valor digitado.

São usados os mesmos operadores de indexação do printf (%d, %f, %c e %s), conforme Tabela 3.

Entrada e Saída de Dados

Função **scanf**: forma geral

```
scanf(string_de_controle,& lista_de_argumentos);
```

Exemplo de **scanf**

```
#include <stdio.h>
/*Exemplo de scanf*/
main()
{
    int num;
    printf("Digite um numero: ");
    scanf("%d", &num);
    printf("%d \n",num);
}
```

Entrada e Saída de Dados

Exemplo com variável string

```
#include <stdio.h>
/*Exemplo da variável string*/
main()
{

char nome[20];
printf("Digite seu nome: ");
scanf("%s", &nome);
printf("\n\n Seu nome e: %s \n", nome);

}
```

Entrada e Saída de Dados

Exemplo

```
#include <stdio.h>
/*Exemplo*/
main()
{

printf ("Teste %% %% \n\n");
printf ("%3f \n\n" , 40.345);
printf ("Um caractere %c e um inteiro %d \n\n",'D',120);
printf ("%s e um exemplo \n\n", "Este");
printf ("%s %d %% \n\n", "Juros de ",10);

}
```

Entrada e Saída de Dados

Exemplo

```
#include <stdio.h>
/*Exemplo*/
main()
{

int dias;                /* Declaracao de Variaveis */
float anos;
printf (“Entre com o numero de dias: ”);
scanf (“%d”, &dias);    /* Entrada de Dados*/
anos = dias/365.0;       /*Conversao Dias->Anos */
printf (“\n\n%d dias equivalem a %f anos.\n”, dias, anos);

}
```

Exercícios de fixação

1) Escreva um código (*script*) que declare 4 variáveis inteiras ao código principal e atribua a essas os valores 10, 20, 30 e 40. Declare 6 variáveis caracteres e atribua a essas as letras 'c','o','e','l','h','a'. Finalmente, o programa deverá imprimir, usando todas as variáveis declaradas.

2) Escreva um código (*script*) que receba os coeficientes de uma função quadrática e retorne:

- $f(x)$
- df/dx
- $f(x)$ para $x=3$
- df/dx para $x=3$

Boas práticas

- 1 Defina o objetivo;
- 2 Faça um fluxograma detalhado;
- 3 Inclua **comentário com cabeçalho** com descritivo do código, autor, data e versão;
- 4 Converta cada passo do **fluxograma** num trecho do código usando a linguagem de programa desejada (“c”);
- 5 **Comente** cada ação por etapa;
- 6 Verifique o fluxo de dados entre etapas para evitar *deadlocks e livelocks*
- 7 Confirme a inserção da biblioteca necessária para as funções, por exemplo `#include<stdio.h>` para **printf** e **scanf**;
- 8 Execute o código.

Exercícios de fixação - resolução

Solução proposta

```
#include<stdio.h>
main() {
//Declarando os números
int num1=10, num2=20, num3=30, num4=40;
//Declarando as letras
char letra1='c', letra2='o', letra3='e';
char letra4='l', letra5='h', letra6='a';
//Imprimindo as variáveis inteiras
printf("As variáveis inteiras são: %d %d %d %d\n\n",
num1, num2, num3, num4);
//Imprimindo as letras
printf("O animal contido é: %c%c%c%c%c%c\n\n",
letra1, letra2, letra3, letra4, letra5, letra6);
}
```

- É possível melhorar esse código?
- Como?
- Quais os impactos?

Operador de atribuição

- A forma geral do operador de atribuição é:
`nome_da_variável = expressão;`
- A expressão pode ser tão simples como uma única constante ou tão complexa quanto você necessite;

Exemplo

```
nome_da_variável = expressão;  
x = 10;  
y = x;
```


Operações matemáticas

Realizam operações matemáticas em relação a um ou mais dados, os operadores com as seguintes estruturas:

Tabela: Operadores matemáticos e funções

| Operador | Ação |
|----------|------------------|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão |
| % | Resto de divisão |
| ++ | Incremento |
| -- | Decremento |

Fonte: Autor,(2020)

Operações matemáticas

Exemplos:

```
int x = 10, y = 20, s;  
s = x + y;
```

Primeiramente,
x é somado com **y**
O resultado é guardado em **s**

```
int x = 10, y = 20, s;  
s = x + 2*y;
```

Primeiramente,
y é multiplicado por **2**
O resultado é somado com **x**
O resultado é guardado em **s**

```
int x = 10, y = 20, s;  
s = 2*(x+y);
```

Primeiramente,
x é somado com **y**
O resultado é multiplicado por **2**
O resultado é guardado em **s**

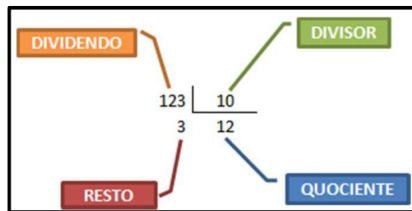
Operações matemáticas

O operador `/` (divisão) quando aplicado a variáveis inteiras, nos fornece o resultado da divisão inteira, ou seja, o resto é truncado;

```
int x=5, y=2; printf("%d", x%y);
```

O operador `%` (módulo) quando aplicado a variáveis inteiras, nos fornece o resto de uma divisão inteira;

Figura: Semântica de operadores na divisão



Fonte: DIAS, (2019)

Operações matemáticas

O operador / (divisão) quando aplicado a variáveis em ponto flutuante nos fornece o resultado da divisão “real”.

```
float x=5, y=2;  
printf(“%f ”, x/y);
```

"Exibe na tela o número 2.500000"

Operações matemáticas

Exemplo

```
#include <stdio.h>
main()
{ int a = 17, b = 3;
  int x, y, w;
  float z = 17.0 , z1, z2;
  x = a / b;
  y = a % b;
  w = z / b;
  z1 = z / b;
  z2 = a / b;
  printf(" x=%d\n y=%d\n w=%d\n z1=%f\n z2=%f\n\n", x,y,w,z1,z2);
  system("pause"); }
```

Recursividade de variáveis

- Processo aplicável quando uma variável é usada para alterar ela mesma;
- Primeiro, as operações à direita da igualdade (“=”) são executadas;
- Em seguida, a variável recebe o seu novo valor.

Recursividade

```
int x = 1100;    // x é declarado como int e recebe valor 1100  
x = x + 2;      // x recebe o seu valor anterior somado a 2;
```

Incrementos/ decrementos e atribuições

Sejam as três expressões comuns em C que envolvem incrementos (decrementos) e atribuições:

- `z = x++;`
- `z = ++x;`
- `z += x;`

Qual a diferença entre elas?

Operador de pós-incremento: $x++$

- Aqui, $x++$ é um operador de pós-incremento.
- Primeiro, o valor de x é atribuído a z .
- Em seguida, x é incrementado em 1.

Exemplo $z = x++$

```
int x = 5;
```

```
int z = x++; // z recebe o valor 5, e x passa a ser 6.
```

Equivalência:

```
z=x;
```

```
x=x+1;
```


Operador de pré-incremento: ++x

- Aqui, ++x é um operador de pré-incremento.
- Primeiro, x é incrementado em 1.
- Em seguida, o novo valor de x é atribuído a z.

Exemplo

```
int x = 5;  
int z = ++x; // x passa a ser 6, e z recebe o valor 6.
```

Equivalência:

```
x=x+1;  
z=x;
```

Operador de atribuição: `+=` **x**

- A expressão `z += x` é equivalente a `z = z + x`.
- Não há incremento de `x` aqui; apenas uma soma.
- O valor de `x` é somado a `z` e o resultado é atribuído a `z`.

Exemplo `z+=x`

```
int z = 5;
```

```
int x = 3;
```

```
z += x; // z passa a ser 8, e x permanece 3.
```

Equivalência:

```
z=z+x;
```

Resumo

| Expressão | Incremento | Valor de z recebe |
|----------------|----------------|---------------------------------------|
| z = x++ | Pós-incremento | Valor de x antes do incremento |
| z = ++x | Pré-incremento | Valor de x após o incremento |
| z += x | Nenhum | z recebe z + x |

Incremento e Decremento

Qual o resultado das variáveis x, y e z depois da seguinte sequência de operações:

Figura: Operações de incremento e decremento

(a) Script

```

1  #include<stdio.h>
2  int main()
3  {
4  int x,y,z;
5  x=y=10;    printf("x=%d y=%d z=%d\n",x,y,z);
6  z=x++;     printf("x=%d y=%d z=%d\n",x,y,z);
7  x=-x;      printf("x=%d y=%d z=%d\n",x,y,z);
8  y++;       printf("x=%d y=%d z=%d\n",x,y,z);
9  x=x+y-z--; printf("x=%d y=%d z=%d\n",x,y,z);
10 }
```

(b) Resultados

```

x= 10 y=10 z= 0
x= 11 y=10 z=10
x=-11 y=10 z=10
x=-11 y=11 z=10
x=-10 y=11 z= 9
```

Fonte: Autor

(*) $z=x++$ é o mesmo que programar $z=x$; $x=x+1$.

Portanto, z recebe o valor atual de x, e x recebe o novo valor $x+1$.

Exemplo: Incremento e Decremento

Exemplo

```
#include <stdio.h>
main() {
    int x, y, z;
    x=y=10;
    printf("x=%d\n y=%d\n\n", x, y);
    z=x++;
    printf("z=%d\n x=%d\n\n", z, x);
    z=++x;
    printf("z=%d\n x=%d\n\n", z, x); x = -x;
    printf("x=%d\n\n", x);
    y++; printf("y=%d\n\n", y);
    system("pause");
}
```

Tabela: Resultados

| Variável | Atribuição |
|----------|------------|
| x = | 10 |
| y = | 10 |
| z = | 10 |
| x = | 11 |
| z = | 12 |
| x = | 12 |
| x = | -12 |
| y = | 11 |

Fonte: Autor,(2020)

Operadores relacionais

Os operadores relacionais do C realizam comparações entre variáveis.

Tabela: Operadores relacionais e função

| Operador | Ação |
|----------|------------------|
| > | Maior que |
| >= | Maior ou igual a |
| < | Menor que |
| <= | Menor ou igual a |
| == | Igual a |
| != | diferente de |

Fonte: DIAS,(2019)

Os operadores acima retornam verdadeiro(1) ou falso(0).

Operadores relacionais

Exemplo

```
#include <stdio.h>
main() {
    int i, j;
    printf("\nEntre com dois números inteiros: \n");
    scanf("%d%d", &i, &j);
    printf("\n\tnum1 = %d e num2 %d\n", i, j);
    printf("\n%d == %d e %d\n", i, j, i==j);
    printf("\n%d != %d e %d\n", i, j, i!=j);
    printf("\n%d <= %d e %d\n", i, j, i<=j);
    printf("\n%d >= %d e %d\n", i, j, i>=j);
    printf("\n%d < %d e %d\n", i, j, i<j);
    printf("\n%d > %d e %d\n\n", i, j, i>j);
    system("pause");
}
```

Exercício

Escreva um programa que declare três variáveis inteiras x , y e z . Seu programa deve solicitar ao usuário os 3 números e armazenar esses números nas variáveis x , y e z .

Realize as operações abaixo e imprima na tela o resultado de X , Y e Z em cada operação:

$$y = x + +$$

$$z = + + y$$

$$x = x - y + z$$

$$y = x - z - -$$

Estruturas de controle

jonatha.costa@ifce.edu.br

Organização

① Linguagem C - Introdução

② Estruturas de Controle - C

- O comando if
- O comando switch
- O comando while
- O comando for

③ Vetores e Matrizes - C

Objetivo da aula

- Estudar as principais estruturas de controle utilizadas em lógica de programação através do uso da Linguagem C.
- São elas:
 - Estrutura sequencial;
 - Estrutura condicional;
 - Estrutura de repetição;
 - Estrutura de Iteração.

Estruturas de controle

- Qual a diferença entre uma estrutura de controle **sequencial e condicional**?
- Qual a diferença entre uma estrutura de controle de **repetição e de iteração**?

Apresente exemplos do dia-a-dia!

Estrutura de Controle Condicional

- Muitos comandos em C contam com um teste condicional que determina o curso da ação;
- Uma expressão condicional chega a um valor verdadeiro ou falso;
- Em C, um valor verdadeiro é qualquer valor diferente de zero, incluindo números negativos;
- O valor falso é 0.

Organização

① Linguagem C - Introdução

Linguagens e conceitos basilares de programação

Linguagem C

Ambientes de Desenvolvimento Integrado

Funções iniciais

② Estruturas de Controle - C

O comando if

O comando switch

O comando while

O comando do-while

O comando for

③ Vetores e Matrizes - C

Estrutura de Controle Condicional: comando if

Sua forma geral é:

if (condição)

```
{  
<comandos se condição verdade>;  
}
```

Exemplo

```
int idade = 19;
```

```
if (idade > 18)
```

```
{  
    printf("Pode tirar carteira de motorista");  
}
```

Estrutura de Controle Condicional

```
#include <stdio.h>
main() {
int num;
printf("Digite um número: "); scanf("%d",&num);
if (num>10) {
printf("\n\n O número é maior que 10! \n");
}
    if (num==10) {
printf("\n\n O número é igual que 10! \n");
}
    if (num<10) {
printf("\n\n O número é menor que 10! \n");
}
}
```

Perceba que cada **if** é independente!

Estrutura de Controle Condicional

Exemplo de *if com else*

```
#include <stdio.h>
main() {
    int num;
    printf("Digite um número: "); scanf("%d",&num);
    if (num>10) {
        printf("\n\n O número é maior que 10! \n");
    }
    else {
        printf("\n\n O número não é maior que 10!! \n");
    }
}
```

Perceba que o **if** está junto ao **else**!

Estrutura de Controle Condicional

Exemplo de *if independente*

```
#include <stdio.h>
main() {
    int num;
    printf("Digite um número: "); scanf("%d",&num);
    if (num>10) {
        printf("\n\n O número é maior que 10! \n");
    }
    else if (num==10) {
        printf("\n\n O número é igual que 10! \n");
    }
    else{
        printf("\n\n O número é menor que 10! \n");
    }
}
```

Perceba que cada **if** está junto ao **else if** e ao **else** !

Exercícios de Estruturas de controle de fluxo - **if**

- 1 Escreva um programa que leia uma nota e verifique se aprovado ou reprovado, considerando a nota de aprovação (7,0);
- 2 Escreva um programa que leia duas notas, calcule a média e verifique se o aluno está aprovado ou reprovado (7,0);
- 3 Escreva um programa que leia duas notas, calcule a média ponderada e verifique se o aluno foi aprovado ou reprovado (7,0). (*Utilize como peso: $nota_1 = 2$ e $nota_2 = 3$*);
- 4 Escreva um programa que leia 5 valores, encontre o maior, o menor e a média utilizando números inteiros;
- 5 Escreva um programa que leia 5 valores, encontre o maior, menor e média utilizando números reais (float).

Organização

① Linguagem C - Introdução

Linguagens e conceitos basilares de programação

Linguagem C

Ambientes de Desenvolvimento Integrado

Funções iniciais

② Estruturas de Controle - C

O comando if

O comando switch

O comando while

O comando do-while

O comando for

③ Vetores e Matrizes - C

Estrutura de Controle Condicional: comando switch

- O comando *switch* testa sucessivamente o valor de uma variável contra uma lista de constantes **inteiras** ou de caracteres;
- É próprio para testar uma variável em relação a diversos valores pré-estabelecidos.
- O comando *break*, faz com que o *switch* seja interrompido assim que uma das declarações seja executada.

Sintaxe do *switch*

```
switch (variavel)
{
  case constante_1:
    declaracao_1;
  break;
  .
  .
  .
  case constante_n:
    delaracao_n;
  break; default:
    declaracao_default;
}
```

Estrutura de Controle Condicional: comando switch

Exemplo

```
#include <stdio.h>
main () {
int num;
printf ("Digite um numero inteiro: "); scanf ("%d", &num);
switch (num)
{
case 9: printf ("\n O numero é igual a 9.\n");
break;
case 10: printf ("\n O numero é igual a 10.\n");
break;
default: printf ("\n O numero nao é nem 9 nem 10.\n");
}
}
```

Estrutura de Controle Condicional: comando switch

Comando switch - Observações

Há duas observações importantes a saber sobre o comando switch:

- switch **só pode testar igualdade**, enquanto que o if pode avaliar uma expressão lógica e/ou relacional;
- Duas constantes **case** no mesmo switch **não podem ter valores idênticos**;

Estrutura de Controle Condicional

Exemplo

```
#include <stdio.h>

main() {
float op1, op2;
char operacao;

printf("Programa solicita dois números e a operação desejada entre eles!\n \n ");
printf("\nDigite o primeiro número: ");
scanf("%f", &op1);
printf("\nDigite o segundo número: ");
scanf("%f", &op2);
printf("\nDigite o operador: (+, -, *, /)");
scanf(" %c", &operacao);
```

parte 1/2

Estrutura de Controle Condicional: comando switch

Exemplo - continuação

```
switch (operacao)
{
case '+': printf("Soma\n%.2f + %.2f é igual a: %.2f\n\n", op1, op2, op1+op2); break;
case '-': printf("Sub.\n%.2f - %.2f é igual a: %.2f\n\n", op1, op2, op1-op2); break;
case '*': printf("Mult.\n%.2f * %.2f é igual a: %.2f\n\n", op1, op2, op1*op2); break;
case '/': printf("Divisao\n%.2f / %.2f é igual a: %.2f\n\n", op1, op2, op1/op2);
break;
default: printf("%c\nOperacao Desconhecida\n\n", operacao);
}
}
```

parte 2/2

Exercícios de Estruturas de Controle Condicional

- 1 Escreva um programa em C para ler uma letra e verificar se é uma vogal ou não;
- 2 Escreva um programa em C que imprima um mês de acordo com o número digitado pelo usuário e informe se o número não tem mês correspondente ou não. (Use o calendário gregoriano);
- 3 Escreva um programa em C que leia um número entre 0 e 10 e escreva este número por extenso. Utilize o comando *switch*.
- 4 Escreva um programa em C que receba um dígito e informe se é uma pontuação identificando o (. : ; ! ?).
- 5 Escreva um programa em C que receba o preço de um produto e o tipo de pagamento. Apresente o preço líquido com desconto de 10% para pagamento à **vista**, 5% para pagamento no **cartão em 1 vez** e acréscimo de 10% se **parcelado**.

Estruturas de Controle de Repetição

Algoritmo com estrutura de repetição

Em alguns casos faz se necessário verificar uma condição dentro do algoritmo; por isso, enquanto o objetivo final não for atingido, o algoritmo repetirá o processo.

- Na linguagem C, comando de iteração (também chamados laços) permitem que um conjunto de instruções seja executado **até que** ocorra uma certa condição;
- As estruturas de repetição em C apresentam-se em 3 formas distintas:
 - while
 - do-while
 - for

Organização

① Linguagem C - Introdução

Linguagens e conceitos basilares de programação

Linguagem C

Ambientes de Desenvolvimento Integrado

Funções iniciais

② Estruturas de Controle - C

O comando if

O comando switch

O comando while

O comando do-while

O comando for

③ Vetores e Matrizes - C

Estruturas de Controle de Repetição

Forma geral do **while**

```
while(condição) {  
    <instruções>;  
}
```

- As **instruções** podem conter comandos simples ou um bloco de funções que se deseja repetir;
- A condição pode ser qualquer expressão;
- O laço se repete quando a condição for verdadeira. Quando a condição é falsa, o controle do programa passa para a linha após o código do laço e, portanto, fora das chaves de fechamento do **while**.

Estruturas de Controle de Repetição while

Comando while - Exemplo

```
#include <stdio.h>
main ()
{
    int num = 0;
    while (num < 100)
    {
        printf ("%d ", num); num++;
    }
}
```

Estruturas de Controle de Repetição while

Comando while - Exemplo

```
#include <stdio.h>
main()
{
    int num1, num2;
    char ch = 's';
    while (ch != 'N' && ch != 'n')
    {
        system("clear");
        printf("\n\nDigite o primeiro número inteiro:"); scanf("%d", &num1);
        printf("\nDigite o segundo número inteiro:"); scanf("%d", &num2);
        printf("\nA soma %d + %d eh: %d", num1, num2, num1+num2);
        printf("\nDeseja realizar uma nova soma? (S/N): ");
        scanf("%c", &ch);
    }
}
```

Estruturas de Controle de Repetição while

Comando while - Exemplo

```
#include <stdio.h>
main() {
int num, count = 1;
printf("Digite um numero: "); scanf("%d", &num);
printf("IMPARES \tPARES\n");
while(count <= num)      {
    if(count%2 == 1)
printf("%d \t", count);
else
printf("\t %d\n",count);
count++;
}
```


Estruturas de Controle de Repetição

Forma geral do comando **do - while**

```
do {  
  comando;  
} while(condição);
```

- O laço do-while verifica a condição ao final do laço;
- Portanto, o laço do-while será executado ao menos uma vez;
- O laço do-while repete até que a condição torne-se falsa.

Estruturas de Controle de Repetição

Diferença entre comando “while” e “do - while”

O comando **do** executa o laço ao menos uma vez, enquanto o comando while somente o faz se a condição inicial da variável de teste for satisfeita!

```
int num = 101;  
do {  
    scanf("%d", &num);  
} while (num<100);
```

```
int num = 101;  
while (num<100) {  
    scanf("%d", &num);  
}
```

Estruturas de Controle de Repetição - while e do-while

- 1 Escreva um programa que mostre todos os números ímpares de 1 até 100;
- 2 Escreva um programa que leia um número e verifica se é um número primo;
- 3 Escreva um programa que solicite um número ao usuário, e mostre sua tabuada completa (de 1 até 10);
- 4 Escreva um programa que solicite 10 números ao usuário, através de um laço while, e ao final mostre qual destes números é o maior;
- 5 Escreva um programa que leia 10 números e escreva a diferença entre o maior e o menor valor lido;
- 6 Escreva um programa que imprima todos os divisores de um número inteiro positivo.

Organização

① Linguagem C - Introdução

Linguagens e conceitos basilares de programação

Linguagem C

Ambientes de Desenvolvimento Integrado

Funções iniciais

② Estruturas de Controle - C

O comando if

O comando switch

O comando while

O comando do-while

O comando for

③ Vetores e Matrizes - C

Estrutura de Iteração: comando "for"

Forma geral do comando "for"

```
for (inicialização; teste; atualização)
{
    sentença(s);
}
```

- Inicialização é, geralmente, um comando de atribuição que é usado para colocar um valor na variável de controle do laço;
- A condição é uma expressão relacional que determina quando o laço acaba;
- O incremento define como a variável de controle do laço varia cada vez que o laço é repetido.

Estrutura de Iteração: comparativo entre os comandos “for” e “while”

Laço “for”

```
main()
{
:   for(inicialização; teste; atualização)
{
sentença(s);
}
:
:
```

Laço “while”

```
main()
{
:   inicialização;
   while(teste)
{
sentença(s);
atualização;
}
:
:
```

Estrutura de Iteração: comparativo entre os comandos “for” e “while”

Imprimir de 0 a 10 com “for”

```
main()
{
int num;
for (num = 0; num <=10; num++)
{
printf(“%d\n”, num);
}
printf(“\n%d\nSaiu do laço
for.\n”,num);
}
```

Imprimir de 0 a 10 com
“while”

```
main()
{
int num;
num=0;
while(num<=10)
{
printf(“%d\n”, num);
num++;
}
printf(“\n%d\nSaiu do
laço while.\n”,num);
}
```

Estrutura de Iteração: laços iteração

Exemplo adicional:

Imprimir de 1 a 100

```
main()
{
  int x;
  for (x = 1; x <=100; x++)
  {
    printf("%d\n", x);
  }
  printf("%d\nSaiu do laço
for.\n",x);
}
```

Imprimir ímpares de 1 a 100

```
main()
{ int i;
  for (i = 1; i <=100; i++)
  {
    if (i%2 == 1)
    {
      printf("%d\n", i);
    }
  }
  printf("%d\nSaiu do laço for.\n",i);
}
```


Estrutura de Iteração: laços“for” aninhados

- Quando um laço for faz parte de outro laço for, dizemos que o laço interno está aninhado.

Imprimir soma x,y de 0 a 2

```
main()
{
int i, j;
for (i = 0; i <3; i++)
{
for (j = 0; j <3; j++)
{ printf("%d\n", i + j);

}
}
}
```

Imprimir de 0 a 99

```
main()
{
int i, j;
for (i = 0; i <= 9; i++)
{
for (j = 0; j <=9; j++)
{ printf("%d%d\n", i,j);

}
}
}
```

Estruturas de Controle de iteração - for

- 1 Escreva um programa que faça uma contagem regressiva de 10 até 1.
- 2 Escreva um programa que leia a idade de 10 pessoas e imprima quantas são maiores de idade;
- 3 Escreva um programa que leia a idade e o peso de 8 pessoas. Calcule e imprima as médias de peso das pessoas da mesma faixa etária e quantas são de cada faixa etária. As faixas são de 1 a 10 anos, de 11-20, de 21-30 e maiores de 30;
- 4 Escreva um programa que calcule o fatorial de um número;
- 5 Escreva um programa que imprima todos os divisores de um número, usando o laço for;
- 6 Escreva um programa que calcule a soma de todos os números pares dos números entre 1 e 100.

Estruturas de Controle de iteração

jonatha.costa@ifce.edu.br

Organização

- 1 Linguagem C - Introdução
- 2 Estruturas de Controle - C
- 3 Vetores e Matrizes - C**

Objetivo da aula

- Estudar a forma como é feita a construção de vetores e matrizes, como também, como é feita a atribuição de valores em cada posição.

Vetores

- Um vetor é uma sequência de vários valores do mesmo tipo, armazenados sequencialmente na memória, e fazendo uso de um mesmo nome de variável para acessar esses valores;
- Cada elemento desta sequência pode ser acessado individualmente através de um índice dado por um número inteiro;
- Os elementos são indexados de 0 até (n-1), onde n é a quantidade de elementos do vetor;
- O vetor tem tamanho fixo durante a execução do programa, definido na declaração.

Vetores

- Durante a execução do *script* não é possível aumentar ou diminuir o tamanho do vetor;
- A tabela a seguir ilustra um vetor com 10 elementos, denominados v_0, v_1, \dots, v_9 todos eles do tipo **int**.

Tabela: Vetor de inteiros com 10 posições

| | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|
| int v[10] | v0 | v1 | v2 | v3 | v4 | v5 | v6 | v7 | v8 | v9 |
|-----------|----|----|----|----|----|----|----|----|----|----|

Fonte: Autor,(2020)

Vetores

Forma geral

```
tipo_da_variável  nome_da_variável
[tamanho];
```

Exemplo

```
int v[10];
float exemplo[20];
```

```
int tamanho=10;
int vetor[tamanho]*2; //Errado!
```

Tabela: Vetores

| |
|-----------|
| int v[10] |
| v0(int) |
| v1(int) |
| v2(int) |
| v3(int) |
| v4(int) |
| v5(int) |
| v6(int) |
| v7(int) |
| v8(int) |
| v9(int) |

Fonte: Autor,(2020)

Vetores

Acesso ao conteúdo de um vetor

Elementos do vetor

vetor[0], vetor[1], vetor[2], ...

Atribuição:

vetor [índice] = valor;

Exemplo

```
int v[10];
v[5] = 3;
v[0] = v[1] + v[2];
```

Tabela: Vetores

| |
|-----------|
| int v[10] |
| v0(int) |
| v1(int) |
| v2(int) |
| v3(int) |
| v4(int) |
| v5(int) |
| v6(int) |
| v7(int) |
| v8(int) |
| v9(int) |

Fonte: Autor,(2020)

Vetores

Exemplo de programação com vetor

Exemplo

```

1  #include <stdio.h>
2  main()
3  {
4  int v[10]={10,9,8,7,6,5,4,3,2,1};
5  int indice;
6      ...
7      for (indice=0; indice<10; indice++)
8      {
9          printf(" %d ", v[indice]);
10         }
11     }

```

Tabela: Vetores

| int v[10] |
|-----------|
| v0(int) |
| v1(int) |
| v2(int) |
| v3(int) |
| v4(int) |
| v5(int) |
| v6(int) |
| v7(int) |
| v8(int) |
| v9(int) |

Fonte: Autor,(2020)

Vetores

Exemplo de programação com vetor

Exemplo

```
1  #include <stdio.h>
2  main() {
3      int valores[10]; int indice;
4      printf("Escreva 10 números inteiros: \n");
5      for(indice=0;  indice<10; indice++)
6          {
7              scanf("%d", &valores[indice]);
8          }
9      printf("Valores em ordem Reversa: \n");
10     for(indice=9; indice>=0; indice - -)
11         {
12             printf("%d \n", valores[indice]);
13         }
14 }
```

Vetores

Declaração de um vetor com conteúdo inicial

Forma geral

tipo vetor[**n**] = {
*elem*₀, *elem*₁, ..., *elem*_{*n*-1} }

Exemplo

int impares[5] = { 1, 3, 5, 7, 9 };

Tabela: Vetores

| |
|-----------|
| int v[10] |
| v0(int) |
| v1(int) |
| v2(int) |
| v3(int) |
| v4(int) |
| v5(int) |
| v6(int) |
| v7(int) |
| v8(int) |
| v9(int) |

Fonte: Autor,(2020)

Vetor de Tamanho Variável

Vetor de tamanho de alocação maior (reserva de espaço)

Exemplo

```
⋮  
int valores[100];           // Tamanho alocado  
int numero_elementos = 10;  // Tamanho usado  
  
int i;  
for (i = 0; i < numero_elementos; i++)  
{  
    printf("%d", valores[i]);  
}  
⋮
```

Vetor de Tamanho Variável

Exemplo

```
1  #include <stdio.h>
2  main()
3  {
4      int valores[100];
5      int numero_valores, i;
6      printf("Quantos valores? (maximo 100)");
7      scanf("%d", &numero_valores);
8      printf("Escreva os numeros: \n");
9      for (i = 0; i < numero_valores; i++)
10     {        scanf("%d", &valores[i]);
11     }
12     printf("Valores em ordem Reversa: \n");
13     for (i = numero_valores-1; i >= 0; i--)
14         {printf("%d \n", valores[i]);
15         }
16 }
```

Exercícios de Vetores

- 1 Escreva um código em C que preencha um vetor com 10 números e indique o maior número ao varrer o vetor preenchido.
- 2 Escreva um código em C que preencha um vetor com 10 números e indique o maior, o menor número e a diferença entre eles.
- 3 Escreva um código em C que preencha um vetor com 10 números e retorne quais são os números ímpares deste vetor.
- 4 Escreva um código em C que preencha um vetor com 10 números e retorne quais são os números primos deste vetor.

Matrizes

MATRIZES

Matrizes

- Uma matriz é uma tabela de valores do mesmo tipo, armazenados sequencialmente e fazendo uso de um mesmo nome de variável para acessar esses valores;
- Cada elemento da tabela pode ser acessado individualmente através de dois índices com valores inteiros. Estes índices poderiam ser interpretados como a linha(i) e a coluna(j) da matriz;
- Os elementos são indexados de 0 até $n-1$, onde n é a quantidade de elementos do vetor;
- A matriz tem tamanho fixo durante a execução do programa, definido na declaração, assim como um vetor.

Matrizes

- A tabela a seguir ilustra uma matriz com 4 linhas e 10 colunas, denominados a_{00} , a_{01} , \dots , a_{39} , todos eles do tipo `int`.

Tabela: Matriz de inteiros `int a[4][10]`

| | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| a_{00} | a_{01} | a_{02} | a_{03} | a_{04} | a_{05} | a_{06} | a_{07} | a_{08} | a_{09} |
| a_{10} | a_{11} | a_{12} | a_{13} | a_{14} | a_{15} | a_{16} | a_{17} | a_{18} | a_{19} |
| a_{20} | a_{21} | a_{22} | a_{23} | a_{24} | a_{25} | a_{26} | a_{27} | a_{28} | a_{29} |
| a_{30} | a_{31} | a_{32} | a_{33} | a_{34} | a_{35} | a_{36} | a_{37} | a_{38} | a_{39} |

Fonte: Autor,(2020)

Matrizes

Declaração de uma matriz

Forma geral

tipo_da_variável nome_da_variável [linhas][colunas];

Exemplo: int a[4][10];

Tabela: Matriz de inteiros int a[4][10]

| | | | | | | | | | |
|----------|----------|----------|----------|--------------|----------|----------|----------|----------|----------|
| a_{00} | a_{01} | a_{02} | a_{03} | a_{04} | a_{05} | a_{06} | a_{07} | a_{08} | a_{09} |
| a_{10} | a_{11} | a_{12} | a_{13} | $a_{14} = 3$ | a_{15} | a_{16} | a_{17} | a_{18} | a_{19} |
| a_{20} | a_{21} | a_{22} | a_{23} | a_{24} | a_{25} | a_{26} | a_{27} | a_{28} | a_{29} |
| a_{30} | a_{31} | a_{32} | a_{33} | a_{34} | a_{35} | a_{36} | a_{37} | a_{38} | a_{39} |

Fonte: Autor,(2020)

Para atribuir o valor 3 na linha 2, coluna 5, escrevemos: $\text{matriz}[1][4] = 3$, pois elemento inicial é **zero**!

Matrizes

Acesso o conteúdo de uma matriz

Elementos da Matriz

`matriz[0][0], matriz[0][1],matriz[0][2] ...`

Atribuição:

`matriz[linha][coluna] = valor;`

Exemplo

```
int matriz [6][10];  
matriz [5][1] = 3;  
matriz [0][2] = matriz [1][0] + matriz [2][3];
```

Matrizes

Exemplo de programação com matrizes

```
1  #include <stdio.h>
2  main()
3  {
4  int lin, col;
5  int matriz[4][10];
6  ...
7  for(lin=0;lin<4;lin++)
8      {
9          for(col=0;col<10;col++)
10             { printf(" %d ",matriz[lin][col]);
11             }
12         }
13     }
14     printf("\n");
14 }
```

Exercícios de matrizes

- 1 Escreva um código em C que preencha uma matriz 3 x 3 e imprima-a.
- 2 Escreva um código em C que crie um algoritmo que leia os elementos de uma matriz inteira de 3 x 3 e imprima outra matriz multiplicando cada elemento da primeira matriz por 2.
- 3 Escreva um código em C que receba 6 valores numéricos inteiros numa matriz 2 x 3 e mostre a soma destes 6 números.
- 4 Escreva um código em C que receba os elementos de uma matriz inteira de 4 x 4 e imprima os elementos da diagonal principal.
- 5 Escreva um código em C que receba os elementos de uma matriz inteira de 3 x 3 e imprima todos os elementos, exceto os elementos da diagonal principal.

Vetores e Matrizes

jonatha.costa@ifce.edu.br

Exercícios

- Veja material auxiliar e lista de códigos em:
<https://github.com/jonathacosta/PL>