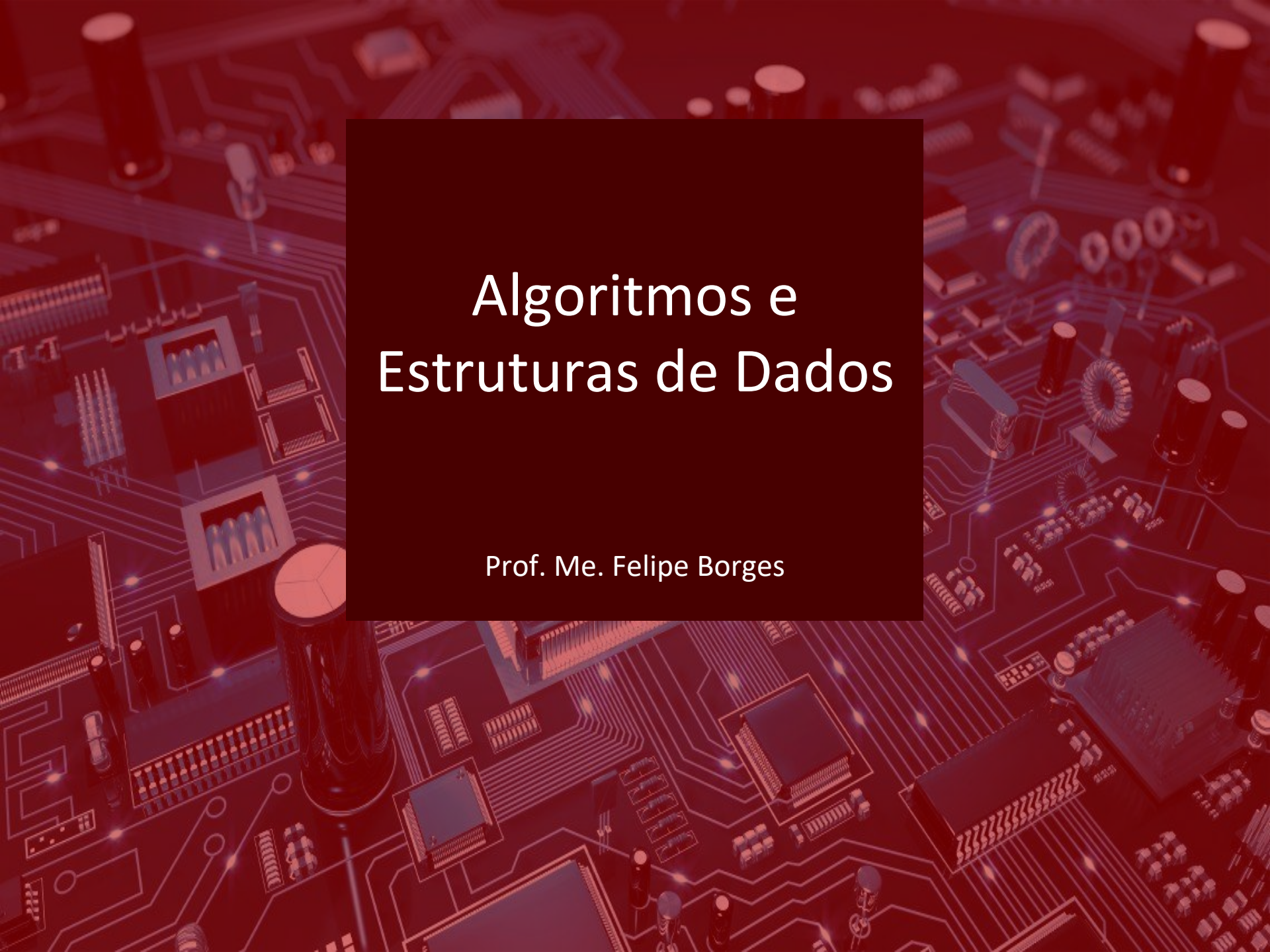


Algoritmos e Estruturas de Dados

Prof. Me. Felipe Borges

Prof. Felipe Borges

Doutorando em Sistemas de Potência – UFMA – Brasil
Mestre em Sistemas de Potência – UFMA – Brasil
MBA em Qualidade e Produtividade – FAENE – Brasil
Graduado em Engenharia Elétrica – IFMA – Brasil
Graduado em Engenharia Elétrica – Fontys – Holanda
Técnico em Eletrotécnica – IFMA – Brasil

Projetos e Instalações Elétricas – Engenharia – Banco do Brasil
Desenvolvimento e Gestão de Projetos – Frencken Engineering BV

Por que laboratório de programação ?

Essa disciplina tem como objetivo prepará-los para as engenharias. Na computação, podemos citar: estrutura de dados, sistemas operacionais, sistemas embarcados, computação gráfica e compiladores.

Por que a linguagem C ?

Uma das linguagens mais utilizadas no desenvolvimento de sistema básico (sistemas operacionais, compiladores, utilitários, drivers, servidores ....), quanto na acadêmia.

Excelente para o estudo de algoritmos e estrutura de dados.

Por que a linguagem C ?

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

Jun 2021	Jun 2020	Change	Programming Language	Ratings	Change
1	1		C	12.54%	-4.65%
2	3	▲	Python	11.84%	+3.48%
3	2	▼	Java	11.54%	-4.56%
4	4		C++	7.36%	+1.41%
5	5		C#	4.33%	-0.40%
6	6		Visual Basic	4.01%	-0.68%
7	7		JavaScript	2.33%	+0.06%

Alguns pontos importantes

- Essa não é uma disciplina de introdução a programação. O foco agora será na linguagem
- Essa é uma disciplina prática, então pressupõe que todos irão realizar as atividades solicitadas e um pouco mais.
- Iremos usar a linguagem C, evitando misturar códigos de C com C++.

Ementa

- Estudo detalhado de uma linguagem de programação.
- Estrutura da linguagem.
- Comandos e declarações.
- Tipos e Representações de dados.
- Mecanismos de entrada e saída de dados.
- Aplicações

Conteúdo programático

1 Introdução a linguagem C
2 Comandos
3 Funções
4 Vetores e Matrizes
5 Ponteiros
6 Strings
7 Alocação dinâmica
8 Noções básicas de Estruturas
9 Noções básicas de Arquivos

IDE - Ambiente de Desenvolvimento Integrado

IDE (do inglês *Integrated Development Environment*) ou **Ambiente de Desenvolvimento Integrado**, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo.

Exemplos de IDE que Geram código para C e C++:

- DEV-C++
- Code::Blocks
- Turbo C

IDE - Ambiente de Desenvolvimento Integrado

As características e ferramentas mais comuns encontradas nos IDEs são:

- Editor: edita o código-fonte do programa escrito nas linguagens suportadas pela IDE;
- Compilador: compila o código-fonte do programa, editado em uma linguagem específica e a transforma em linguagem de máquina;
- Depurador (*debugger*): auxilia no processo de encontrar e corrigir defeitos no código-fonte do programa, na tentativa de aprimorar a qualidade de software;

Linguagens de programação

Linguagens de programação são conhecimentos escritos e formais que seguem um conjunto de instruções e regras para o desenvolvimento de softwares.

Linguagens de programação

Um programa de computador é um conjunto de instruções que representam um algoritmo para a resolução de algum problema. Estas instruções são escritas através de um conjunto de códigos (símbolos e palavras).

Este conjunto de códigos possui regras de estruturação lógica e sintática própria. Diz-se que este conjunto de símbolos e regras formam uma linguagem de programação.

Tradução

- MONTADOR (*assembler*)
  - Tradutor para linguagens.
- COMPILADOR:
  - Traduz todo o programa de uma vez.
- INTERPRETADOR:
  - Traduz o programa instrução por instrução.

Estrutura básica de um programa C

```
diretivas para o pré-processador
declaração de variáveis globais
main ()   {
    declaração de variáveis locais da função main
    comandos da função main
}
```

Diretivas para o processador - Bibliotecas

- Diretiva `#include` permite incluir uma biblioteca
- Bibliotecas contêm funções pré-definidas, utilizadas nos programas
- Exemplos

`#include <stdio.h>`	Funções de entrada e saída
`#include <stdlib.h>`	Funções padrão
`#include <math.h>`	Funções matemáticas
`#include <system.h>`	Funções do sistema
`#include <string.h>`	Funções de texto

O ambiente Dev-C++

- O Dev-C++ é um ambiente de desenvolvimento de programas em C e C++ com editor, compilador, bibliotecas e debugger
- Pode ser baixado de https://sourceforge.net/projects/orwelldevcpp/
- Vamos criar apenas programas na linguagem **C**

Usando o Dev-C++

- Inicie o Dev-C++ pelo ícone ou pelo menu
- Crie um novo arquivo, com o comando *File, New Source File*
- Edite o programa da página seguinte

Usando o Dev-C++

```
#include <stdio.h>

main(){

   printf ("Alo mundo!");

   system(“pause");
}
```

Usando o Dev-C++

- Salve o programa com o nome **exemplo.c.** Para tanto, selecione o menu File, Save unit as
- Compile o programa com o comando **Executar**, **Compilar** ou com a tecla **F9**
- Se houver algum erro de sintaxe, aparece uma ou mais mensagens no rodapé da janela. Neste caso, corrija o programa e repita.
- Se não houver erros, execute o programa com o comando **Executar**, **Executar** ou com a tecla **F10**

Dicas

- **Termine todas as linhas com ;**
- Sempre salve o programa antes de compilar
- **Sempre compile o programa antes de executar**
- Quando ocorrer um erro de compilação, dê um duplo clique sobre a mensagem de erro para destacar o comando errado no programa
- Verifique também a linha anterior, que pode ser a responsável pelo erro, especialmente se faltar o **;**
- **Use comentários, iniciados por //**

Template

```
#include <stdio.h>
main()
{
   printf ("Alo mundo!");
   system("pause");
}
```

Declarações

- Declaram as variáveis e seus tipos
- Os nomes das variáveis devem conter apenas letras, dígitos e o símbolo _
- Até 32 caracteres
- Os principais tipos são: **int**, **float**, e **char**
- Exemplos

```
int n;
int quantidade_valores;
float x, y, somaValores;
char sexo;
char nome[40];
```

Linguagem C diferencia letras maiúsculas de minúsculas!

int n, N;

n é diferente de N!

Exemplo

Real: n1, n2, n3, media

```
#include <stdio.h>
main()
{
    real n1, n2, n3, media;




system(“pause");
}
```

Comando de atribuição

- Atribui o valor da direita à variável da esquerda
- O valor pode ser uma constante, uma variável ou uma expressão
- Exemplos

```
x = 4;  --> lemos: x recebe 4
y = x + 2;
y = y + 4;
valor = 2.5;
```

Entrada e Saída

- Função **scanf**

```
scanf ("formatos", &var1, &var2,...)
```

Exemplos:

```
int i, j;
float x;
char c;
char nome[10];
scanf("%d", &i);
scanf("%d %f", &j, &x);
scanf("%c", &c);
scanf("%s", nome);
```

%d	**inteiro**
%f	**float**
%lf	**double**
%c	**char**
%s	**palavra**

Exemplo

Real: n1, n2, n3, media

ler n1, n2, n3

```
#include <stdio.h>
main()
{
    float n1, n2, n3, media;
    scanf ("%f %f %f",&n1, &n2, &n3);



system(“pause");
}
```

Operadores Matemáticos

Operador	Exemplo	Comentário
+	`x + y`	Soma x e y
-	`x - y`	Subtrai y de x
*	`x * y`	Multiplica x e y
/	`x / y`	Divide x por y
% Ou **mod** no portugol	`x % y`	Resto da divisão de x por y
++	`x++`	Incrementa em 1 o valor de x
--	`x--`	Decrementa em 1 o valor de x

Exemplo

Real: n1, n2, n3, media

ler n1, n2, n3

media=(n1+n2+n3)/3

```
#include <stdio.h>
main()
{
    float n1, n2, n3, media;
   scanf ("%f %f %f",&n1, &n2, &n3);
   media=(n1+n2+n3)/3;



system(“pause");
}
```

Entrada e Saída

- Função **printf**

```
printf ("formatos", var1, var2,...)
```

Exemplos:

```
int i, j;
float x;
char c;
char nome[10];
printf("%d", i);
printf("%d, %f", j, x);
printf("%c", c);
printf("%s", nome);
```

%d	inteiro
%f	float
%lf	double
%c	char
%s	palavra

Exemplo

Real: n1, n2, n3, media

ler n1, n2, n3

media=(n1+n2+n3)/3

exibir media

```
#include <stdio.h>
main()
{
    real n1, n2, n3, media;
    scanf ("%lf %lf %lf",&n1, &n2, &n3);
    media=(n1+n2+n3)/3;
    printf (“%f”,media);

    system(“pause");
}
```

Exemplo

```
#include <stdio.h>
main()
{
    real n1, n2, n3,
    scanf ("%f %f %f",
          &n1, &n2, &n3);

    media=(n1+n2+n3)/3;
    printf (“%f”,media);


    system("PAUSE");
}
```

```
#include <stdio.h>
main()
{
    float n1, n2, n3, media;
    printf(“Digite 3 notas: ”);
    scanf ("%f %f %f",&n1, &n2, &n3);
    media=(n1+n2+n3)/3;
    printf (“A média é %0.2f”,media);

}
```

Exercício

1) Tendo como dados de entrada a altura de uma pessoa, construa um programa que calcule (e mostre) seu peso ideal, utilizando a seguinte fórmula:
   - peso ideal = (72.7*altura) - 58

Funções Matemáticas

Função	Exemplo	Comentário
`ceil`	`ceil(x)`	Arredonda o número real para cima; ceil(3.2) é 4
`cos`	`cos(x)`	Cosseno de x (x em radianos)
`exp`	`exp(x)`	**e** elevado à potencia x
`fabs`	`fabs(x)`	Valor absoluto de x
`floor`	`floor(x)`	Arredonda o número para baixo; floor(3.2) é 3
`log`	`log(x)`	Logaritmo natural de x
`log10`	`log10(x)`	Logaritmo decimal de x
`pow`	`pow(x, y)`	Calcula x elevado à potência y
`sin`	`sin(x)`	Seno de x
`sqrt`	`sqrt(x)`	Raiz quadrada de x
`tan`	`tan(x)`	Tangente de x

```
#include <math.h>
```

Operadores Relacionais

Operador	Exemplo	Comentário
==	x == y	O conteúdo de x é igual ao de y
!=	x != y	O conteúdo de x é diferente do de y
<=	x <= y	O conteúdo de x é menor ou igual ao de y
>=	x >= y	O conteúdo de x é maior ou igual ao de y
<	x < y	O conteúdo de x é menor que o de y
>	x > y	O conteúdo de x é maior que o de y

Operadores Lógicos

- **&&** (E lógico): retorna verdadeiro se ambos os operandos são verdadeiros e falso nos demais casos.
  Exemplo: if(a>2 && b<3).

- **||** (OU lógico): retorna verdadeiro se um ou ambos os operandos são verdadeiros e falso se ambos são falsos.
  Exemplo: if(a>1 || b<2).

- **!** (NÃO lógico): usada com apenas um operando. Retorna verdadeiro se o operando é falso e vice-versa.
  Exemplo: if(!var).

break;

- Como no Python, o comando break sai imediatamente do loop mais interno em que ela é encontrada. Naturalmente, a declaração tem um ponto e vírgula a seguir.

continue;

- Também como em Python, o comando continue pula para a parte inferior do loop mais interno no qual ele é encontrado e testa se deve repetir o loop novamente. Tem um ponto e vírgula também.

Operadores Lógicos

Tabela E	Tabela OU	Tabela NÃO
V e V → V	V ou V → V	Não V → F
V e F → F	V ou F → V	Não F → V
F e V → F	F ou V → V	
F e F → F	F ou F → F	

Estrutura condicional simples

- Comando **if**

```
if (condição)
    comando;

if (condição) {
    comando1;
    comando2;
    comando3;
}
```

```
if (a<menor)
   menor=a;


if (a<menor) {
    menor=a;
    printf ("%d", menor);
}
```

em pseudo-código:

```
se (a<menor) entao menor=a;
```

Estrutura condicional composta

- Comando **if...else**

```
if (condição)
    comando;
else
    comando;

if (condição) {
    comando1;
    comando2;
} else {
    comando3;
    comando4;
}
```

```
if (peso==peso_ideal){
   printf ("Vc esta em forma!");
}
else{
   printf ("Necessario fazer dieta!");
}
```

em pseudo-código:

```
se (peso=peso_ideal) entao
      exibir "Vc está em forma!"
senao
            exibir "Necessário fazer dieta!"
fimse
```

Exemplo

```
#include <stdio.h>
main()
{
    real n1, n2, n3,
    scanf ("%f %f %f",
          &n1, &n2, &n3);

    media=(n1+n2+n3)/3;
    printf (“%f”,media);


    system("PAUSE");
}
```

```
#include <stdio.h>
main()
{
    float n1, n2, n3, media;
    printf(“Digite 3 notas: ”);
    scanf ("%f %f %f",&n1, &n2, &n3);
    media=(n1+n2+n3)/3;
    printf (“A média é %0.2f”,media);

    if(media>=7){
        printf(“Voce esta aprovado,
    com a media %f”,media);
    }
    else{
        printf(“Voce esta reprovado”);
    }
}
```

Exercício

2) Faça um programa que leia um número inteiro e mostre uma mensagem indicando se este número é positivo ou negativo.

3) Explique porque está errado fazer if (num=10) ... O que irá acontecer?

Exercício

2)

```c
#include<stdio.h>
main(){
    int num;
    printf("Digite um \t numero \n");
    scanf("%d",&num);

    if(num>=0){
        printf("O numero eh positivo");
    }
    else{
        printf("O numero eh negativo");
    }
}
```

switch (conceito) { case 'A':...

O comando switch não tem equivalente em Python, mas é essencialmente equivalente a uma forma particular de uma declaração if... elif... elif... else onde cada um dos testes é para valores diferentes da mesma variável.

Um comando switch é útil quando você tem vários blocos de código possíveis, um dos quais deve ser executado com base no valor de uma expressão específica. Aqui está uma instância clássica da declaração switch:

```
switch (letter_grade) {
case 'A':
   gpa =gpa+ 4;
   credits =credits+1;
   break;
case 'B':
   gpa += 3;
   credits += 1;
   break;
case 'C':
   gpa += 2;
   credits += 1;
   break;
case 'D':
   gpa += 1;
   credits += 1;
   break;
case 'W':
   break;
default:
   credits += 1;
}
```

switch (conceito) { case 'A':...

switch (conceito) { case 'A':...

Dentro dos parênteses após a palavra chave switch, temos uma expressão cujo valor deve ser um caractere ou inteiro. O computador avalia essa expressão e desce para uma das palavras-chave case com base em seu valor. Se o valor é o caractere A, então o primeiro bloco é executado (gpa + = 4; créditos += 1;); se for B, então o segundo bloco é executado; se não for nenhum dos caracteres (como um F), o bloco que segue a palavra-chave default é executado.

switch (conceito) { case 'A':...

O comando break no final de cada bloco é um detalhe crucial: Se o comando break for omitida, o computador continua no bloco seguinte. Em nosso exemplo acima, se omitimos todas as declarações break, então uma nota de A levaria o computador a executar não apenas o caso A, mas também os casos B, C, D, W e default. O resultado seria que o gpa aumentaria em 4 + 3 + 2 + 1 = 10, enquanto que credits aumentaria em 5.

Estrutura de repetição

- Comando **for**

```
for (var=valor inicial; condição; incremento)
    comando;

for (var=valor inicial; condição; incremento)
{
    comando1;
    comando2
    comando3;
}
```

Exemplo:

```
for (cont=3; cont<=11; cont++) {
    printf ("%d",cont);
}
```

Pseudo-código:

Para CONT = 3 **até** 11 **repetir**
Mostrar CONT

Exercícios

4) Faça um programa que escreva de 50 a 100.

Exercícios

4) Faça um programa que escreva de 50 a 100.

```
1  #include<stdio.h>
2  main(){
3      int cont;
4      for(cont=50;cont<=100;cont++){
5          printf("%d",cont);
6      }
7  }
```

Estrutura de repetição

- Comando **while**

```
while (condição)
    comando;

while (condição) {
    comando1;
    comando2
    comando3;
}
```

Exemplo:

```
while (N != 0) {
   scanf ("%d",&N);
}
```

Pseudo-código:

MAIOR = 0
N = 1
Enquanto (N <> 0) **repetir**
Ler N

Exercício

5) Faça um programa que conte de 1 a 10 usando o comando **while**.

Exercício

5) Faça um programa que conte de 1 a 10 usando o comando **while**.

Exercício

5) Faça um programa que conte de 1 a 10 usando o comando **while**.

```c
#include<stdio.h>
main(){
    int cont;
    cont=1;
    while(cont<=10){
        printf("%d, ",cont);
        cont=cont+1;
        //cont++;
    }
}
```

Estrutura de repetição

- Comando **do...while**

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

do {
    comando1;
    comando2
    comando3;
} while (condição);
```

Exemplo:

```
cont=0;
do {
    cont = cont + 1;
    printf("%d \n",cont);
} while (cont < 10);
```

Em pseudo-código:

CONTADOR = 0
Repetir
CONTADOR = CONTADOR + 1
exibir CONTADOR
enquanto CONTADOR < 10

Exercício

6) Faça um programa que conte de 1 a 10 usando o comando **do...while**.

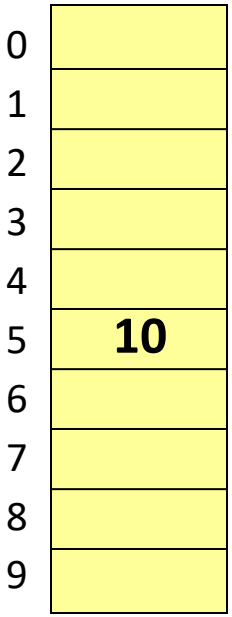
Vetores (array)

- Trata-se de automatizar a declaração de um grande número de dados de um mesmo tipo simples. As variáveis assim declaradas se acessam através de um índice de tipo int.

- Declaração:
  - int v[100];
  - primeira posição =0;
  - última posição=99;
- Atribuição:
  - v [9] = 87;
- Acessar um valor:
  - a = v[9];

Vetores (array)

- int v[10];



V[5]=10;

printf (“%d”,V[5]);

Funções

- São estruturas que permitem ao programador separar o código do seu programa em blocos.
- Uma função tem a seguinte forma :

```
tipo_de_retorno Nome_da_funcao (parâmetros){
  /*corpo da função */
}
```

Funções que não retornam valor :

```
#include <stdio.h>
void ehPar (int x){
    if (x % 2) {
        printf ("O numero nao eh par!\n");
    }
    else {
        printf ("O numero eh par!\n");
    }
}
int main(){
    ehPar (3);
    system("pause");
}
```

Funções que retornam valor

```
#include <stdio.h>
#include <stdlib.h>
int ehPar (int x){
// o operador % retorna o resultado da divisão por 2
   if ((x % 2)==0) {
       return 0;
}
       else{
       return 1;
       }
}
int main(){
   int i = ehPar (5);
       if (i!=0){
      printf ("O numero eh impar! \n");
   }
   else{
          printf ("O numero eh par! \n");
       }
   system("pause");
}
```

Funções

- Em C, uma função deve ser declarada acima do local onde você a usa. No programa C de [Figura a seguir], definimos a função gcd () primeiro, depois a função main ().

- Isto é significativo: Se nós trocamos as funções gcd () e main (), o compilador iria reclamar em main() que a função gcd () não foi declarada. Isso porque C assume que um compilador lê um programa de cima para baixo: No momento em que chega ao main (), ele não foi informado sobre uma função gcd (), e por isso ele acredita que a função não exista.

Funções

- Isso gera um problema, especialmente em programas maiores que abrangem vários arquivos, em que as funções de um arquivo precisarão chamar funções em outro. Para contornar isso, C fornece a noção de um protótipo de função, onde escrevemos o cabeçalho da função mas omitimos a definição do corpo.

Funções

- Por exemplo, digamos que queremos quebrar nosso programa C em dois arquivos: O primeiro arquivo, math.c, conterá a função gcd () e o segundo arquivo, main.c, conterá o main () função. O problema com isso é que, na compilação main.c, o compilador não saberá sobre a função gcd () que está tentando chamar.

Funções

- Uma solução é incluir um protótipo de função em main.c.

```
int gcd(int a, int b);

int main() {
    printf("GCD: %d\n", gcd(24, 40));
    return 0;
}
```

Funções

- A linha int gcd ... é o protótipo da função. Você pode ver que começa da mesma forma que uma definição de função, mas nós simplesmente colocamos um ponto-e-vírgula onde o corpo da função normalmente seria. Ao fazer isso, estamos declarando que a função será eventualmente definida, mas ainda não a definimos. O compilador aceita isso e obedientemente compila o programa sem reclamações.

Funções

- A linha int gcd ... é o protótipo da função. Você pode ver que começa da mesma forma que uma definição de função, mas nós simplesmente colocamos um ponto-e-vírgula onde o corpo da função normalmente seria. Ao fazer isso, estamos declarando que a função será eventualmente definida, mas ainda não a definimos. O compilador aceita isso e obedientemente compila o programa sem reclamações.

Operadores de Atribuição

Operador	Exemplo	Comentário
=	x = y	Atribui o valor de y a x
+=	x += y	Equivale a x = x + y
-=	x -= y	Equivale a x = x – y
*=	x *= y	Equivale a x = x * y
/=	x /= y	Equivale a x = x / y
%=	x %= y	Equivale a x = x % y

Tipo Char

É tipo similar a um inteiro, mas de apenas 1 byte, ou seja, permite armazenar 256 valores distintos e usa a tabela ASCII para representar cada carácter do teclado.

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	“	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	‘	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Tipo Char

Por exemplo:

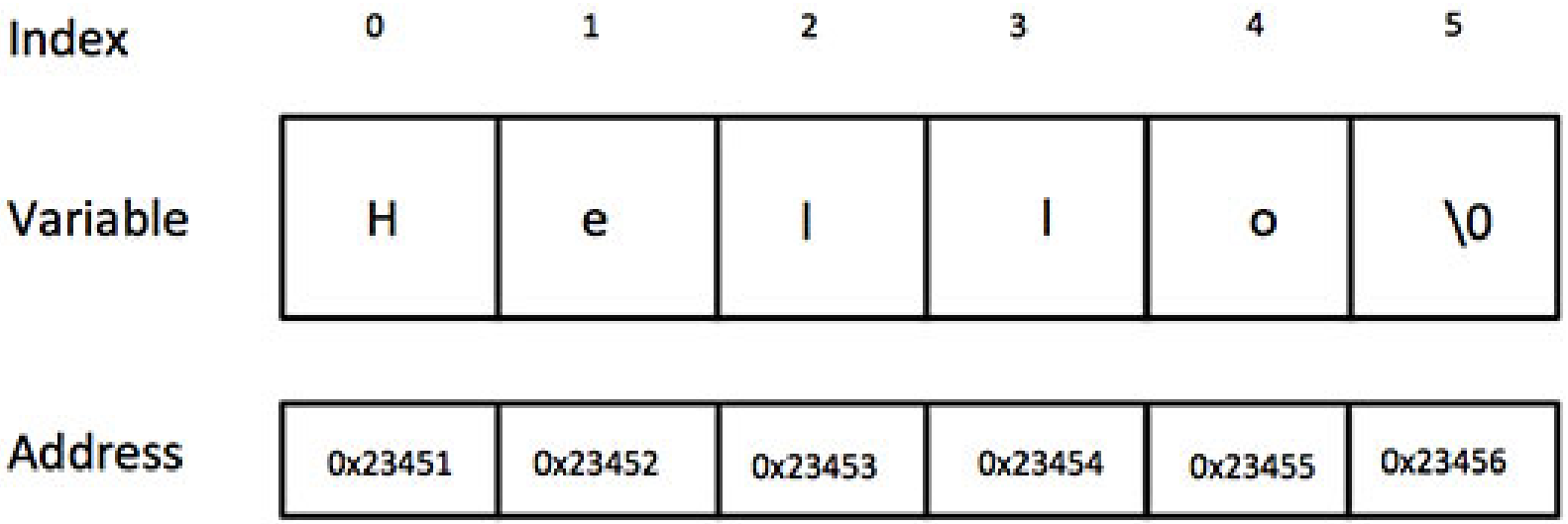
```
int main () {

    char letra1 = 65;
    char letra2 = 'A';  // letra1 == letra2
}
```

Array de caractere

Logo, a linguagem usa um array para poder apresentar o que é denominado de string.

Organização interna

- - vetor do tipo char, terminado pelo caractere nulo ('\0')
- - é reservada uma posição adicional no vetor para o caractere de fim da cadeia



Inicialização

A linguagem suporta o uso de aspas duplas para representar uma string, e nesse caso o caractere nulo é representado implicitamente.

```
int main ( void )
{
   char cidade[ ] = "Rio";
   printf("%s \n", cidade);
   return 0;
}
```

≡

```
int main ( void )
{
   char cidade[ ]={'R', 'i', 'o', '\0'};
   printf("%s \n", cidade);
   return 0;
}
```

Inicialização

Exemplos:

```
char s1[] = "";
char s2[] = "Rio de Janeiro";
char s3[81];
char s4[81] = "Rio";
```

Inicialização

```
char s1[] = "";
char s2[] = "Rio de Janeiro";
char s3[81];
char s4[81] = "Rio";
```

- s1 é uma cadeia de caracteres vazia (vetor com 1 elemento que armazena apenas o caractere '\0');

- s2 é uma cadeia de 14 caracteres armazenada em vetor com 15 elementos, inicializada com o conteúdo declarado e terminada com '\0';

Inicialização

```
char s1[] = "";
char s2[] = "Rio de Janeiro";
char s3[81];
char s4[81] = "Rio";
```

- s3 pode armazenar uma cadeia com até 80 caracteres em um vetor com 81 elementos (já que uma cadeia de caracteres sempre tem que ser terminada com o caractere ‘\0’, que ocupa uma posição a mais no vetor).

- s3 não foi inicializada;

Inicialização

```
char s1[] = "";
char s2[] = "Rio de Janeiro";
char s3[81];
char s4[81] = "Rio";
```

- s4 pode armazenar uma cadeia com até 80 caracteres em um vetor com 81 elementos, mas apenas os quatro primeiros elementos são inicializados com 'R', 'i', 'o' e '\0', respectivamente.

Leitura

Com o scanf usamos o especificador %s, e não precisa usar o operador de endereço já que é um vetor.

```
char cidade[81];
...
scanf("%s", cidade);
...
```

Leitura

O %s lê apenas palavras, caso encontre um espaço ele termina a leitura. Para capturar a linha fornecida pelo usuário até que ele tecle “Enter” podemos escrever:

```
char cidade[81];
scanf(" %80[^\n]", cidade);
```

O %80 indica que pode ter no máximo 80 caracteres

Strings

- Não existe um tipo String em C.
- Strings em C são uma array do tipo char que termina com ‘\0’.
- Para literais String, o próprio compilador coloca ‘\0’.

```
#include <stdio.h>
main(){
   char re[] = "lagarto";
   printf ("%s", re);
   system("pause");
}
```

Para ler uma String

- Comando gets

```
#include <stdio.h>
main(){
   char re [80];
   printf ("Digite o seu nome: ");
   gets(re);
   printf ("Oi %s\n", re);
   system("pause");
}
```

Funções

Como string não são nativas, precisamos usar funções específicas. Não podemos usar operadores relacionais e comandos de atribuição como nos outros tipos de dados.

Funções: strlen

A função strlen retorna o número de caracteres em uma cadeia de caracteres. O caractere '\0' não é contado.

```
#include<stdio.h>
#include<string.h>
void main(void){
    char nome[80]="Joao da Silva";
    printf("%s contem %d caracteres",nome,strlen(nome));
}
```

Funções: strcpy

A função strcpy é usada para copiar o conteúdo de uma cadeia de caracteres (fonte) para outra cadeia (destino). Usaremos ela ao invés da simples atribuição.

```
#include<stdio.h>
#include<string.h>
void main(void) {
    char nome[]="Joao da Silva",aluno[50];
    strcpy(aluno,nome);
    printf("O nome do aluno e %s",aluno);
}
```

Funções: strcmp

A função strcmp é usada para fazer a comparação lexicográfica de duas cadeias. Se usarmos operadores relacionais, estaríamos comparando valores dos ponteiros

- Se as cadeias são iguais, isto é, se str1 e str2 têm mesmo comprimento e caracteres iguais nos elementos de mesmo índice, a função retorna 0
- Se str1 for lexicograficamente maior do que str2, a função retorna 1
- Se str2 for lexicograficamente maior do que str1, a função retorna -1

Funções: strcmp

A função strcmp é usada para fazer a comparação lexicográfica de duas cadeias.

```
#include<stdio.h>
#include<string.h>
void main(void)
{
char nome1[]="Joao da Silva", nome2[]="Maria Fernanda";
if(!strcmp(nome1, nome2))
    printf("%s e igual a %s", nome1, nome2);
else if(strcmp(nome1, nome2)>0)
    printf("%s vem depois de %s",nome1,nome2);
else
    printf("%s vem depois de %s",nome2,nome1);
}
```

Funções: strcat

A função strcat anexa o conteúdo de uma cadeia de caracteres (fonte) ao final de outra cadeia (destino). O tamanho da cadeia de destino deve ser suficiente para armazenar o total de caracteres das duas cadeias:

```
#include<stdio.h>
#include<string.h>
void main(void)
{
    char nome[30]="Joao", sobreNome[]=" da Silva";
    strcat(nome, sobreNome);
    printf("O nome do aluno e %s", nome);
}
```

Funções

Para mais detalhes, veja em:
https://www.cplusplus.com/reference/cstring/

Exercícios – Faça em Linguagem C

1) Transforme um valor em dólar, para reais. Sendo q um dólar vale R$ 5,27.

Exercícios – Faça em Linguagem C

2. O Shopping da Ilha agora usa uma nova tarifação para o estacionamento, mostrado na tabela abaixo:

TEMPO	VALOR
ATÉ 15 MINUTOS	GRÁTIS
ATÉ 3 HORAS	R$ 8,00
A CADA HORA ADICIONAL (ACIMA DAS 3 PRIMEIRAS)	R$ 2,00

O programa recebe o tempo em minutos e diz quanto o usuário precisa pagar.

Exercícios – Faça em Linguagem C

3. O usuário digita três lados de um triângulo e o programa diz se os valores formam ou não um triângulo.

Obs: Pela regra, para se formar um triângulo, cada lado tem que ser menor que a soma dos outros dois lados. Ou seja (C < A+B), (B< A+C) e (A< B+C).

Exercícios – Faça em Linguagem C

4) Leia o nome, número de horas trabalhadas e número de dependentes de um funcionário. Após a leitura, escreva qual o Nome, salário bruto, os valores descontados para cada tipo de imposto e finalmente qual o salário líquido do funcionário. Considerando que:

a) A empresa paga R$12 por hora e R$40 por dependentes.

b) Sobre o salário são descontados 8,5% p/ o INSS e 5% p/ IR

Exercícios – Faça em Linguagem C

5) Leia uma distância em km, o preço da gasolina em reais e exiba quantos litros de gasolina o carro irá consumir e quanto será gasto em reais. Considere que o carro faz 12 km/l de gasolina.

Exercícios – Faça em Linguagem C

6) Faça um algoritmo que calcule e escreva o valor a ser pago a sua provedora de acesso à Internet. Para isso você deverá ler a quantidade de horas que você utilizou. Sabe-se que você pagará R$ 30,00 por até 72 horas de uso (valor básico), caso você tenha usado mais de 72 horas, então deve ser acrescido mais 5% no valor básico para cada hora extra utilizada.

Exercícios – Faça em Linguagem C

7) Leia um código de votação e escreva a ordem de classificação e o percentual de votos de cada candidato. Considere: a) F = fim da eleição; b) X,Y,Z = códigos dos candidatos; c) N = voto nulo e d) B = voto em branco.