Visão geral da Linguagem C

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará
Campus Quixadá

2º semestre/2025

Tópicos

- Compilação e execução
- Elementos básicos da linguagem C
- Estruturas de seleção e repetição
- Vetores
- Structs e enumerações
- Ponteiros
- Alocação dinâmica de memória
- Ponteiros para ponteiros
- Arrays de caracteres

Tutoriais online para estudo rápido e consulta

- learn-c.org: Um tutorial interativo gratuito e sem necessidade de instalação: você escolhe o capítulo e já pode programar no navegador
- W3Schools C Tutorial: Plataforma gratuita com editor, você modifica o código e executa direto no site
- FreeCodeCamp: Apresenta uma abordagem direta e eficiente com os principais conceitos da linguagem

Apresentação da Linguagem

- A linguagem C é considerada uma linguagem de programação **imperativa** e **procedural**
  - Imperativa: O programador descreve passo a passo como o computador deve executar as instruções
  - Procedural: O código é organizado em funções (procedimentos), que podem ser reutilizadas. Segue os princípios da **programação estruturada** (sequência, seleção e repetição).

Apresentação da Linguagem

- A linguagem C é considerada uma linguagem de programação **imperativa** e **procedural**
  - Imperativa: O programador descreve passo a passo como o computador deve executar as instruções
  - Procedural: O código é organizado em funções (procedimentos), que podem ser reutilizadas. Segue os princípios da **programação estruturada** (sequência, seleção e repetição).

Foi desenvolvida por Dennis Ritchie (com contribuições de Ken Thompson) nos Laboratórios Bell, nos anos 1970.

É uma das linguagens mais influentes da história da computação.

Apresentação da Linguagem

Padronizações da linguagem:

- K&R C (1978, não oficial)
- ANSI C (C89, 1989)
- ISO C (C90, 1990)
- C95 (1995)
- C99 (1999)
- **C11 (2011)**
- C17/C18 (2017/2018)
- C23 (2023/2024)

C possui diversas bibliotecas

- assert.h
- ctype.h
- errno.h
- float.h
- limits.h
- math.h
- stdlib.h
- string.h
- time.h
- stddef.h
- iso646.h
- locale.h
- setjmp.h
- signal.h
- stdarg.h
- stdbool.h
- stdint.h
- stdio.h
- uchar.h
- wchar.h
- wctype.h

Primeiro Programa — Hello World

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h>

int main() {
    printf("Hello world!\n");
    return 0;
}
```

Primeiro Programa — Hello World

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello world!\n");
5     return 0;
6 }
```

Supondo que o programa acima esteja no arquivo `programa.cpp`, para compilar no terminal do Linux e depois executar:

- `$ gcc -Wall -Wextra -pedantic-errors programa.c -o main`
- `$ ./main`
  - `-Wall` é a abreviação de "warn all" – ativa (quase) todos os avisos que o g++ pode lhe informar.
  - `-Wextra` ativa alguns avisos extras que não são ativados por `-Wall`
  - `-pedantic-errors` rejeita programas que não seguem o padrão ISO
  - `-o` altera o nome do arquivo de saída.

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Elementos básicos da linguagem C

Elementos básicos da linguagem

Como em outras linguagens:

- Comentários de código
- Variáveis e Constantes
- Identificadores
- Tipos Fundamentais
- Representação numérica
- Vetores, strings, ponteiros, estruturas e enumerações
- Estruturas de controle de fluxo
- Funções
- entre outras...

Comentários

- Um comentário é uma descrição inserida diretamente no código-fonte do programa e que é ignorada pelo compilador. Serve apenas para uso do programador.
- C permite fazer comentários de duas maneiras diferentes: por linha ou por bloco.

```c
#include <stdio.h>

int main() {
  /* --- Exemplo de comentario em bloco ---
  A funcao printf
  serve para
  escrever na tela
  */
  printf("Hello World\n"); // Um comentario em linha

  return 0;
}
```

Variáveis

Variáveis

- Em computação, uma variável é uma posição de memória onde poderemos guardar determinado dado ou valor e modificá-lo ao longo da execução do programa.

Variáveis

- Em computação, uma variável é uma posição de memória onde poderemos guardar determinado dado ou valor e modificá-lo ao longo da execução do programa.
- Quando criamos uma variável e armazenamos um valor dentro dela, o computador reserva um espaço associado a um endereço de memória onde podemos guardar o valor dessa variável.

Variáveis

- Em computação, uma variável é uma posição de memória onde poderemos guardar determinado dado ou valor e modificá-lo ao longo da execução do programa.
- Quando criamos uma variável e armazenamos um valor dentro dela, o computador reserva um espaço associado a um endereço de memória onde podemos guardar o valor dessa variável.
- O nome da variável é chamado identificador.

Variáveis

- Em computação, uma variável é uma posição de memória onde poderemos guardar determinado dado ou valor e modificá-lo ao longo da execução do programa.
- Quando criamos uma variável e armazenamos um valor dentro dela, o computador reserva um espaço associado a um endereço de memória onde podemos guardar o valor dessa variável.
- O nome da variável é chamado identificador.

```
1 #include <stdio.h>
2
3 int main() {
4   int x;  //declara a variavel mas nao define o valor
5   printf("x = %d\n", x);;
6   x = 5;  //define o valor de x como sendo 5
7   printf("x = %d\n", x);
8   return 0;
9 }
```

Inicialização de variáveis escalares

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog07.c

int main() {
  int z = 23; // por atribuicao -- assignment

  printf("Valor de z: %d\n", z);

  int k = 45.89; // ok, mas com perda de precisao

  printf("Valor de k: %d\n", k);

  return 0;
}
```

Identificadores

A linguagem C estipula algumas regras para a escolha dos identificadores:

- Um identificador é um conjunto de caracteres que podem ser letras, números ou *underscores* (_).

Identificadores

A linguagem C estipula algumas regras para a escolha dos identificadores:

- Um identificador é um conjunto de caracteres que podem ser letras, números ou *underscores* (_).
- O identificador deve sempre iniciar com uma letra ou o *underscore* (_).

Identificadores

A linguagem C estipula algumas regras para a escolha dos identificadores:

- Um identificador é um conjunto de caracteres que podem ser letras, números ou *underscores* (_).
- O identificador deve sempre iniciar com uma letra ou o *underscore* (_).
- A linguagem C é case-sensitive, ou seja, uma palavra escrita utilizando caracteres maiúsculos é diferente da mesma palavra escrita com caracteres minúsculos.

Identificadores

A linguagem C estipula algumas regras para a escolha dos identificadores:

- Um identificador é um conjunto de caracteres que podem ser letras, números ou *underscores* (_).
- O identificador deve sempre iniciar com uma letra ou o *underscore* (_).
- A linguagem C é case-sensitive, ou seja, uma palavra escrita utilizando caracteres maiúsculos é diferente da mesma palavra escrita com caracteres minúsculos.
- Palavras reservadas não podem ser usadas como nome de variáveis.

Identificadores

A linguagem C estipula algumas regras para a escolha dos identificadores:

- Um identificador é um conjunto de caracteres que podem ser letras, números ou *underscores* (_).
- O identificador deve sempre iniciar com uma letra ou o *underscore* (_).
- A linguagem C é case-sensitive, ou seja, uma palavra escrita utilizando caracteres maiúsculos é diferente da mesma palavra escrita com caracteres minúsculos.
- Palavras reservadas não podem ser usadas como nome de variáveis.
- As palavras reservadas formam a sintaxe da linguagem e possuem funções específicas.

Palavras reservadas da linguagem C

A partir do padrão C11 existem 44 palavras reservadas

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
inline	int	long	register
restrict	return	short	signed
sizeof	static	struct	switch
typedef	union	unsigned	void
volatile	while	_Alignas	_Alignof
_Atomic	_Bool	_Complex	_Generic
_Imaginary	_Noreturn	_Static_assert	_Thread_local

Tipos nativos da linguagem C (64 bits)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Tipo	Bytes	Faixa de valores
char / signed char	1	-128 ... 127
unsigned char	1	0 ... 255
short	2	-32.768 ... 32.767
unsigned short	2	0 ... 65.535
int	4	-2.147.483.648 ... 2.147.483.647
unsigned int	4	0 ... 4.294.967.295
long	8	-9.223e18 ... 9.223e18
unsigned long	8	0 ... 1.844e19
long long	8	-9.223e18 ... 9.223e18
unsigned long long	8	0 ... 1.844e19
float	4	$\pm$1.175e-38 ... $\pm$3.403e38
double	8	$\pm$2.225e-308 ... $\pm$1.798e308
long double	16	depende da implementação

Operador `sizeof`

Podemos exibir o número de bytes requeridos por um tipo na nossa máquina usando o operador **`sizeof()`**:

```c
#include <stdio.h> // prog05.c

int main() {
    printf("char: %ld\n", sizeof(char));
    printf("short: %ld\n", sizeof(short));
    printf("int: %ld\n", sizeof(int));
    printf("long: %ld\n", sizeof(long int));
    printf("float: %ld\n", sizeof(float));
    printf("double: %ld\n", sizeof(double));
    printf("long double: %ld\n", sizeof(long double));
}
```

Conversão de tipos em C

- Conversão de tipo é o processo de conversão de um valor de um tipo de dados para outro tipo.
- Há dois modos de se converter tipos em C:

Conversão de tipos em C

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Conversão de tipo é o processo de conversão de um valor de um tipo de dados para outro tipo.
- Há dois modos de se converter tipos em C:
  - Conversão Implícita ou Coerção: em que o compilador transforma automaticamente um tipo de dados fundamental em outro.
  - Conversão Explícita (Casting): em que o programador usa um operador de *casting* para direcionar a conversão.

Conversão de tipos Implícita

Existem dois tipos de conversão implícita:

- Promoção Numérica: Quando um valor de um tipo é implicitamente convertido para um tipo semelhante maior.

$$\text{char} \rightarrow \text{short} \rightarrow \text{int} \rightarrow \text{long} \rightarrow \text{float} \rightarrow \text{double}$$

  Promoções não resultam em perda de dados.

Conversão de tipos Implícita

Existem dois tipos de conversão implícita:

- Promoção Numérica: Quando um valor de um tipo é implicitamente convertido para um tipo semelhante maior.

  $$\text{char} \rightarrow \text{short} \rightarrow \text{int} \rightarrow \text{long} \rightarrow \text{float} \rightarrow \text{double}$$

  Promoções não resultam em perda de dados.

- Rebaixamento Numérico: Quando um valor de um tipo é implicitamente convertido para um tipo semelhante menor.

  $$\text{char} \leftarrow \text{short} \leftarrow \text{int} \leftarrow \text{long} \leftarrow \text{float} \leftarrow \text{double}$$

  Rebaixamentos numéricos resultam em perda de dados.

Conversão Implícita — Exemplo


```c
#include <stdio.h> // prog29.c

int main() {
  // Promocao numerica
  int v = 30;
  float f = v;
  printf("%f\n", f/7); // Imprime o valor 4.28571
  printf("%d\n", v/7); // Imprime o valor 4
```

Conversão Implícita — Exemplo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog29.c

int main() {
  // Promocao numerica
  int v = 30;
  float f = v;
  printf("%f\n", f/7); // Imprime o valor 4.28571
  printf("%d\n", v/7); // Imprime o valor 4

  // Rebaixamento Numerico
  int a = 35.0/4.0;

  printf("%d\n", a); // Imprime o valor 8

  return 0;
}
```

Conversão de tipos Explícita (casting)

- No exemplo abaixo, gostaríamos que a variável `f` recebesse o valor 2.5. No entanto, ela recebe o valor 2.

```
1 int i1 = 10;
2 int i2 = 4;
3 float f = i1 / i2;
```

Conversão de tipos Explícita (casting)

- No exemplo abaixo, gostaríamos que a variável `f` recebesse o valor 2.5. No entanto, ela recebe o valor 2.

```
1 int i1 = 10;
2 int i2 = 4;
3 float f = i1 / i2;
```

- Para avisarmos o compilador que queremos usar a divisão de ponto flutuante em vez da divisão de números inteiros, usamos um operador de conversão de tipos (cast).

Conversão de tipos Explícita (casting)

- No exemplo abaixo, gostaríamos que a variável `f` recebesse o valor 2.5. No entanto, ela recebe o valor 2.

```
1 int i1 = 10;
2 int i2 = 4;
3 float f = i1 / i2;
```

- Para avisarmos o compilador que queremos usar a divisão de ponto flutuante em vez da divisão de números inteiros, usamos um operador de conversão de tipos (cast).

Exemplo de casting:

Conversão de tipos Explícita (casting)

- No exemplo abaixo, gostaríamos que a variável `f` recebesse o valor 2.5. No entanto, ela recebe o valor 2.

```
1 int i1 = 10;
2 int i2 = 4;
3 float f = i1 / i2;
```

- Para avisarmos o compilador que queremos usar a divisão de ponto flutuante em vez da divisão de números inteiros, usamos um operador de conversão de tipos (cast).

Exemplo de casting:

```
1 int i1 = 10;
2 int i2 = 4;
3 float f = (float) i1 / i2;
```

Constantes

Uma constante é uma variável especial que permite guardar determinado dado na memória do computador, com a certeza de que ele não se alterará durante a execução do programa.

Constantes

Uma constante é uma variável especial que permite guardar determinado dado na memória do computador, com a certeza de que ele não se alterará durante a execução do programa.

A fim de declarar uma constante, basta colocar a palavra-chave `const` antes do tipo de variável:

Constantes

Uma constante é uma variável especial que permite guardar determinado dado na memória do computador, com a certeza de que ele não se alterará durante a execução do programa.

A fim de declarar uma constante, basta colocar a palavra-chave **const** antes do tipo de variável:

```
1 #include <stdio.h>
2
3 int main() {
4     const double pi = 3.14159;
5     // pi = 3.14; // Erro de compilacao
6     printf("pi = %f\n", pi);
7     return 0;
8 }
```

Constantes — Exemplo 2

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog04a.c

int main() {
    int n;

    printf("Digite o tamanho do array: ");
    scanf("%d", &n);

    const int MAX = n; // definida em tempo de execucao

    int array[MAX];

    for(int i = 0; i < MAX; i++) {
        printf("array[%d] = %d\n", i, array[i]);
    }

  return 0;
}
```

Constantes

O C suporta dois tipos de constantes:

- Constantes em tempo de compilação: são aquelas cujos valores de inicialização podem ser resolvidos em tempo de compilação. Exemplo: variável `pi` do exemplo anterior.

Constantes

O C suporta dois tipos de constantes:

- Constantes em tempo de compilação: são aquelas cujos valores de inicialização podem ser resolvidos em tempo de compilação. Exemplo: variável `pi` do exemplo anterior.
  - Constantes deste tipo permitem que o compilador realize otimizações.

Constantes

O C suporta dois tipos de constantes:

- Constantes em tempo de compilação: são aquelas cujos valores de inicialização podem ser resolvidos em tempo de compilação. Exemplo: variável `pi` do exemplo anterior.
  - Constantes deste tipo permitem que o compilador realize otimizações.

- Constantes em tempo de execução: são aquelas cujos valores de inicialização só podem ser resolvidos em tempo de execução. Exemplo: variável `MAX`, do exemplo anterior.

Escrevendo na tela

Função printf

- A função `printf()` é uma das funções de escrita de dados de C.

Função printf

- A função `printf()` é uma das funções de escrita de dados de C.
- Seu nome vem da expressão em inglês print formatted, ou seja, escrita formatada.

Função printf

- A função `printf()` é uma das funções de escrita de dados de C.
- Seu nome vem da expressão em inglês print formatted, ou seja, escrita formatada.
- Basicamente, a função `printf()` escreve na saída de vídeo (tela) um conjunto de valores, caracteres e/ou sequência de caracteres de acordo com o formato especificado.

Função printf

- A função `printf()` é uma das funções de escrita de dados de C.
- Seu nome vem da expressão em inglês print formatted, ou seja, escrita formatada.
- Basicamente, a função `printf()` escreve na saída de vídeo (tela) um conjunto de valores, caracteres e/ou sequência de caracteres de acordo com o formato especificado.
- A forma geral da função `printf()` é:

```
printf(‘‘tipos de saída’’, lista de variáveis);
```

Exemplo: Escrevendo uma mensagem de texto

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog100.c

int main() {
    printf("Esse texto sera escrito na tela.\n");
    return 0;
}
```

Escrevendo valores formatados

- Quando queremos escrever dados formatados na tela, usamos a forma geral da função, a qual possui os **tipos de saída** ou **especificador de formato**.

Escrevendo valores formatados

- Quando queremos escrever dados formatados na tela, usamos a forma geral da função, a qual possui os **tipos de saída** ou **especificador de formato**.
- Cada tipo de saída é precedido por um sinal de %, e um tipo de saída deve ser especificado para cada variável a ser escrita.

Escrevendo valores formatados

- Quando queremos escrever dados formatados na tela, usamos a forma geral da função, a qual possui os **tipos de saída** ou **especificador de formato**.
- Cada tipo de saída é precedido por um sinal de %, e um tipo de saída deve ser especificado para cada variável a ser escrita.
- se quiséssemos escrever uma única expressão com o comando `printf()`, faríamos:

```
printf(‘‘%tipo_de_saida’’, expressao);
```

Escrevendo valores formatados

- Quando queremos escrever dados formatados na tela, usamos a forma geral da função, a qual possui os **tipos de saída** ou **especificador de formato**.
- Cada tipo de saída é precedido por um sinal de %, e um tipo de saída deve ser especificado para cada variável a ser escrita.
- se quiséssemos escrever uma única expressão com o comando `printf()`, faríamos:

```
printf(‘‘%tipo_de_saida’’, expressao);
```

- Se fossem duas as expressões a serem escritas, faríamos:

```
printf(‘‘%tipo1’’, ‘‘%tipo2’’, expressao1, expressao2);
```

Especificadores de Formato do `printf()`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Especificador	**Tipo de dado impresso**
%d, %i	int (decimal com sinal)
%u	unsigned int (decimal sem sinal)
%o	unsigned int (octal)
%x, %X	unsigned int (hexadecimal, min/maiúsculo)
%f	double ou float (ponto flutuante decimal)
%F	double ou float (ponto flutuante decimal, maiúsculo)
%c	int (caractere único, convertido de `unsigned char`)
%s	char* (string terminada em \0)
%p	void* (endereço de ponteiro)
%%	imprime o caractere %

Exemplo 1: Escrevendo valores na tela

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog101.c

int main() {
    printf("%d\n", 55);           // int
    printf("%i\n", 234);          // int
    printf("%f\n", 55.56f);       // float
    printf("%f\n", 77.654321);    // double
    printf("%F\n", 77.654321);    // double
    printf("%c\n", 67);           // char
    printf("%s\n", "ana maria"); // char*
    printf("%%\n");
    return 0;
}
```

Modificadores de Comprimento do `printf()`

Modificador	**Tipo de dado correspondente**
h	`short int` ou `unsigned short int`
l	`long int`, `unsigned long int`
ll	`long long int` ou `unsigned long long int`
z	`size_t`
t	`ptrdiff_t`
L	`long double`

- Os modificadores de comprimento que podem ser usados junto com os especificadores de formato do `printf()` no C.

Exemplo 2: Escrevendo valores na tela

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog102.c

int main() {
    short v = 87;
    long z = 2356432;
    size_t x = 7654;
    long long w = 87654321;
    long double ldo = 771.65432132189;
    printf("%hd\n", v);     // short int
    printf("%ld\n", z);     // long int
    printf("%zu\n", x);     // size_t
    printf("%lld\n", w);    // long long int
    printf("%Lf\n", ldo);   // long double
    return 0;
}
```

Caracteres de Escape em C

Sequência	**Significado**
\n	Nova linha (line feed)
\t	Tabulação horizontal
\v	Tabulação vertical
\b	Backspace
\r	Retorno de carro (carriage return)
\f	Form feed (avanço de página)
\a	Alerta sonoro (bell)
\0	Caractere nulo (terminador de string)
\'	Aspas simples
\"	Aspas duplas
\?	Ponto de interrogação literal
\\	Barra invertida

Exemplo 3: caracteres de escape

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog103.c

int main() {
    printf("\?\n");              // ?
    printf("\'atilio\'\n");      // 'atilio'
    printf("\"ana\"\n");         // "ana"
    printf("\\lucas\n");         // \lucas
    printf("an\0a maria");       // an
    printf("\n");
    printf("oi,\tcomo vai?\n");
    printf("oi\vcomo\vvai\vvocê?\n");
    printf("claro\bisse\n");
    return 0;
}
```

Width e Alinhamento no `printf()`

Tipo	**Comando**	**Saída**
Inteiro	`%5d`	Alinhado à direita, campo mínimo 5
Inteiro	`%-5d`	Alinhado à esquerda, campo mínimo 5
Inteiro	`%*d`	Width dinâmica (p.ex. 8):
Float	`%10.2f`	Campo 10, 2 casas decimais, direita
Float	`%-10.2f`	Campo 10, 2 casas decimais, esquerda
String	`%5s`	Alinhado à direita, campo 5
String	`%-5s`	Alinhado à esquerda, campo 5
String	`%-*s`	Width dinâmica

Exemplo de largura e alinhamento

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog110.c

int main() {
    int x = 87;

    printf("|%5d|\n", x);
    printf("|%-5d|\n", x);
    printf("|%*d|\n", 10, x);
    printf("|%10.2f|\n", 3.14);
    printf("|%-10.2f|\n", 3.14);
    printf("|%5s|\n", "Oi");
    printf("|%-5s|\n", "Oi");
    printf("|%-*s|\n", 8, "Oi");
}
```

A função `putchar()`

- A função `putchar()` (**put character**) permite escrever um único caractere na tela. Sua forma geral é:

```
int putchar(int caractere);
```

A função `putchar()`

- A função `putchar()` (**put character**) permite escrever um único caractere na tela. Sua forma geral é:

```
int putchar(int caractere);
```

- Essa função recebe como parâmetro de entrada um único valor inteiro. Esse valor será convertido em caractere e mostrado na tela.

A função `putchar()`

- A função `putchar()` (**put character**) permite escrever um único caractere na tela. Sua forma geral é:

```
int putchar(int caractere);
```

- Essa função recebe como parâmetro de entrada um único valor inteiro. Esse valor será convertido em caractere e mostrado na tela.

- Se NÃO ocorrer erro, a função retorna o próprio caractere que foi escrito. Caso contrário, a constante **EOF** (definida na biblioteca `stdio.h`) é retornada.

Exemplo: putchar()


```c
#include <stdio.h> // prog104.c

int main() {
    char c = 'a';
    int x = 65;

    putchar(c);          // escreve 'a'
    putchar('\n');       // quebra de linha
    putchar(x);          // escreve o caractere 65
    putchar('\n');       // quebra de linha

    return 0;
}
```

Exemplo: putchar()


```c
#include <stdio.h> // prog104.c

int main() {
    char c = 'a';
    int x = 65;

    putchar(c);          // escreve 'a'
    putchar('\n');       // quebra de linha
    putchar(x);          // escreve o caractere 65
    putchar('\n');       // quebra de linha

    return 0;
}
```

- a conversão em `putchar()` é direta no momento da impressão, ou seja, o valor 65 é convertido no caractere ASCII correspondente, no caso, o caractere A.
- Além disso, o comando `putchar()` também aceita o uso de sequências de escape, como o caractere $\backslash n$.

Lendo dados do teclado

A função `scanf()`

- A função `scanf()` é uma das funções de entrada/leitura de dados da linguagem C. Seu nome vem da expressão em inglês **scan formatted**.

A função `scanf()`

- A função `scanf()` é uma das funções de entrada/leitura de dados da linguagem C. Seu nome vem da expressão em inglês **scan formatted**.

- A função `scanf()` lê do teclado um conjunto de valores, caracteres e/ou sequência de caracteres de acordo com o formato especificado. A forma geral da função é:

```
scanf(‘‘tipos de entrada’’, lista de variáveis);
```

A função `scanf()`

- A função `scanf()` é uma das funções de entrada/leitura de dados da linguagem C. Seu nome vem da expressão em inglês **scan formatted**.

- A função `scanf()` lê do teclado um conjunto de valores, caracteres e/ou sequência de caracteres de acordo com o formato especificado. A forma geral da função é:

```
scanf(‘‘tipos de entrada’’, lista de variáveis);
```

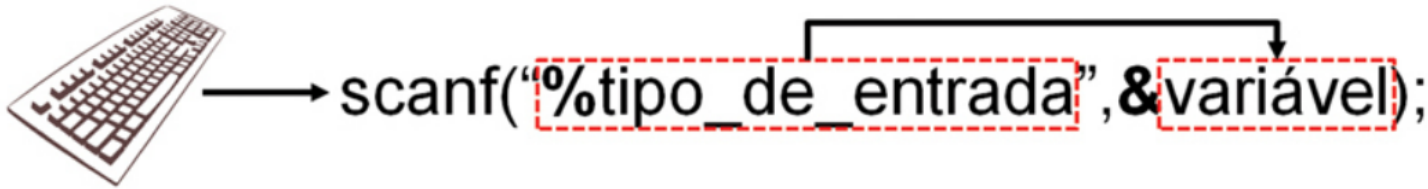
- Os **tipos de entrada** especificam o formato de entrada dos dados que serão lidos pela função `scanf()`. Cada tipo de entrada é precedido por um sinal de %, e um tipo de entrada deve ser especificado para cada variável a ser lida.

A função `scanf()`

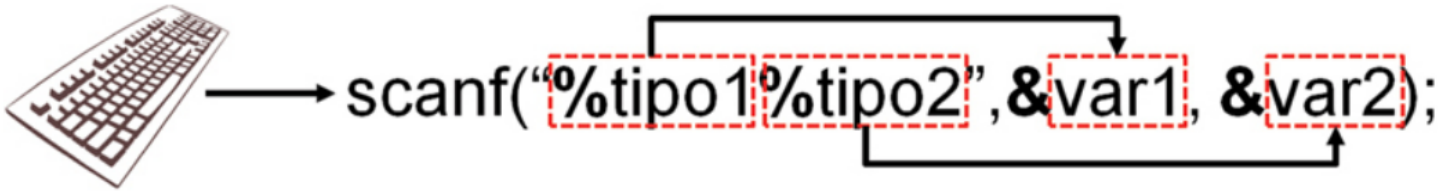
- se quiséssemos ler uma única variável com o comando `scanf()`, faríamos:

A função `scanf()`

- se quiséssemos ler uma única variável com o comando `scanf()`, faríamos:



- Se fossem duas as variáveis a serem lidas, faríamos:



Exemplo 1: scanf()

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog105.c

int main() {
    int x, z;
    float y;
    double k;

    // Leitura de dois inteiros
    scanf("%d%d", &x, &z);

    // Leitura de um float
    scanf("%f", &y);

    // Leitura de um double
    scanf("%lf", &k);

    printf("x=%d, z=%d, y=%f, k=%f\n", x, z, y, k);

    return 0;
}
```

Exemplo 2: scanf()

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog106.c

int main() {
    int dia, mes, ano;

    printf("Digite a data de hoje ");
    printf("no formato dd/mm/aaaa: ");

    scanf("%d/%d/%d", &dia, &mes, &ano);

    printf("data digitada: %d/%d/%d\n", dia,mes,ano);

    return 0;
}
```

A função `getchar()`

- A função `getchar()` (**get character**) permite ler um único caractere do teclado. Sua forma geral é:

```
int getchar(void);
```

A função `getchar()`

- A função `getchar()` (**get character**) permite ler um único caractere do teclado. Sua forma geral é:

```
int getchar(void);
```

- Se NÃO ocorrer erro a função `getchar()` o código ASCII do caractere lido. Se ocorrer erro: a constante EOF (definida na biblioteca `stdio.h`) é retornada.

Exemplo 1: getchar()

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog107.c

int main() {
    char nome[35];
    printf("Digite o seu primeiro nome: ");

    scanf("%s", nome);

    getchar();  // lê o \n que fica no buffer

    printf("seu nome é: %s\n", nome);

    printf("digite algo para encerrar...");

    getchar();

    return 0;
}
```

Exemplo 2: getchar()


```c
#include <stdio.h> // prog108.c

int main() {
    int idade;
    printf("Digite a sua idade: ");

    scanf("%d", &idade);

    getchar();  // lê o \n que fica no buffer

    printf("sua idade é: %d\n", idade);

    printf("digite algo para encerrar...");

    getchar();

    return 0;
}
```

Exemplo 3: getchar()

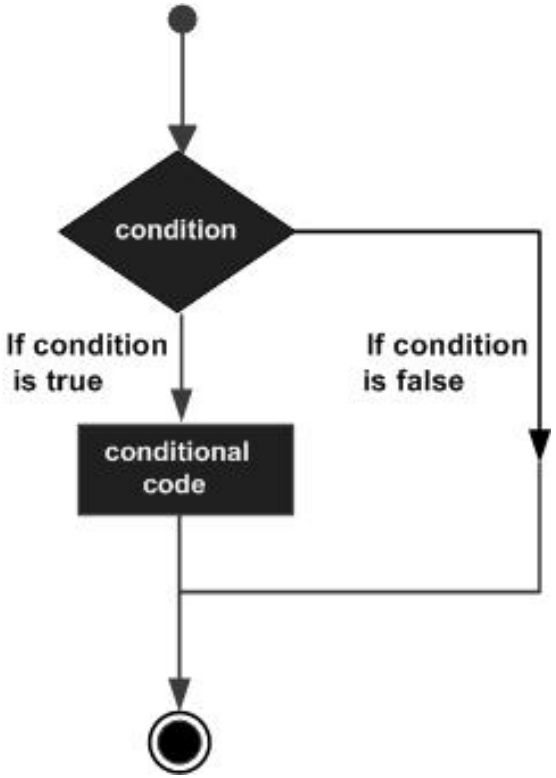
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog108.c

int main() {
    int idade;
    printf("Digite a sua idade: ");
    scanf("%d", &idade);

    char nome[35];
    printf("Digite o seu primeiro nome: ");
    scanf("%s", nome);

    printf("sua idade é: %d\n", idade);
    printf("seu nome é: %s\n", nome);
    printf("digite algo para encerrar...");

    getchar();  // lê o \n que fica no buffer após scanf()
    getchar();

    return 0;
}
```

Estruturas de Seleção

Estruturas de Seleção — If.. else

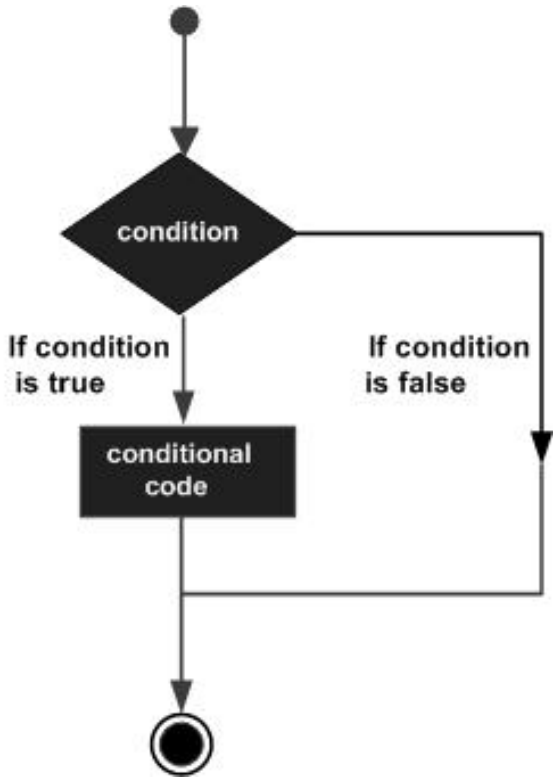
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ




```c
int c;
scanf("%d", &c);

if (c == 1) { // If
    printf("igual a 1");
}
```

Estruturas de Seleção — If.. else

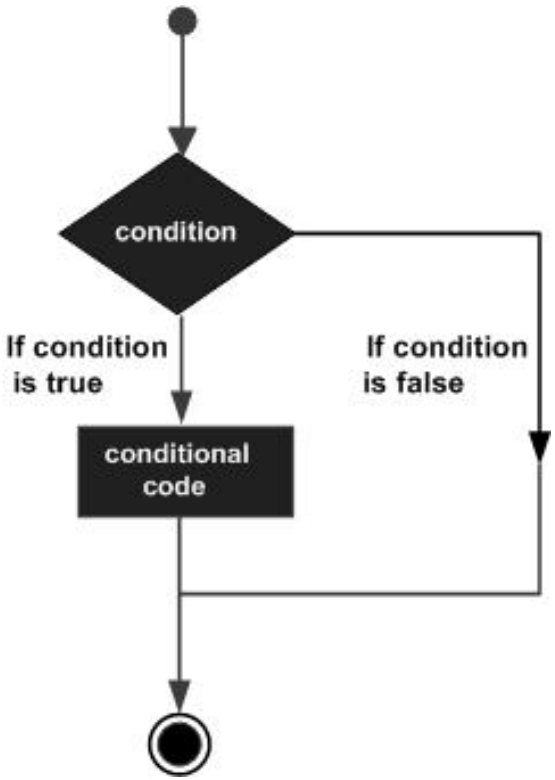
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ




```c
int c;
scanf("%d", &c);

if (c == 1) { // If
    printf("igual a 1");
}

if (c == 2) { // If .. else
  printf("igual a 2");
} else {
  printf("nao eh 1 nem 2");
}
```

Estruturas de Seleção — If.. else

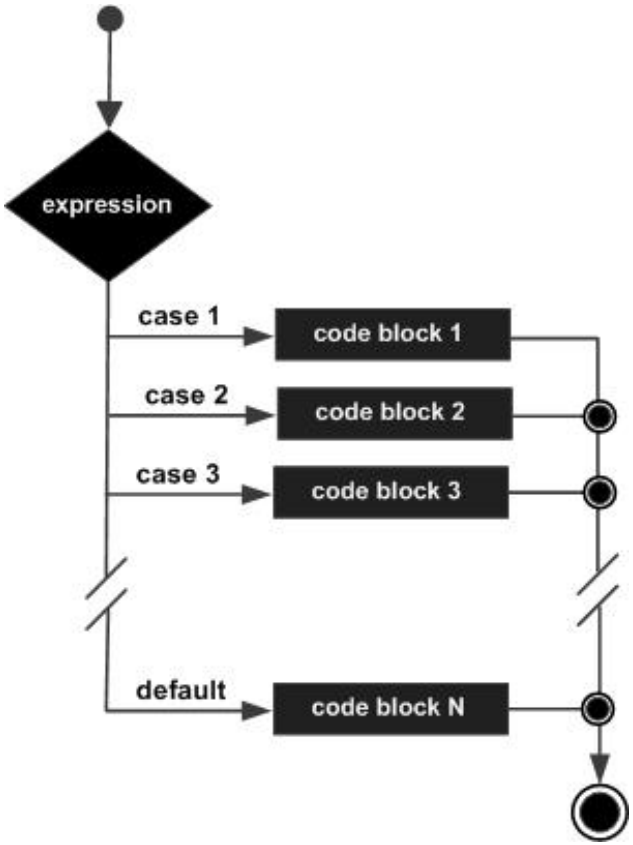
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ




```c
int c;
scanf("%d", &c);

if (c == 1) { // If
    printf("igual a 1");
}

if (c == 2) { // If .. else
  printf("igual a 2");
} else {
  printf("nao eh 1 nem 2");
}

if (c == 3) { // If else encadeado
  printf("igual a 3");
} else if (c == 4) {
  printf("igual a 4");
} else {
  printf("nao eh 3 nem 4");
}
```

Estruturas de Seleção — Switch

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ




```c
int c;
scanf("%d", &c);

switch (c) // int,char,short,long
{
    case 1:
        printf("1 \n");
        break;
    case 2:
        printf("2 \n");
        break;
    case 3:
        printf("3 \n");
        break;
    case 4:
        printf("4 \n");
        break;
    default:
        printf("5 \n");
        break;
}
```

Alternativa para o if–else

Operador ternário

O operador ternário avalia uma condição de teste e executa um bloco de código com base no resultado da condição.

Sua sintaxe é:

```
condition ? expression1 : expression2 ;
```

- se `condition` for `true` então `expression1` é executado;
- se `condition` for `false` então `expression2` é executado.

Operador ternário – Exemplo

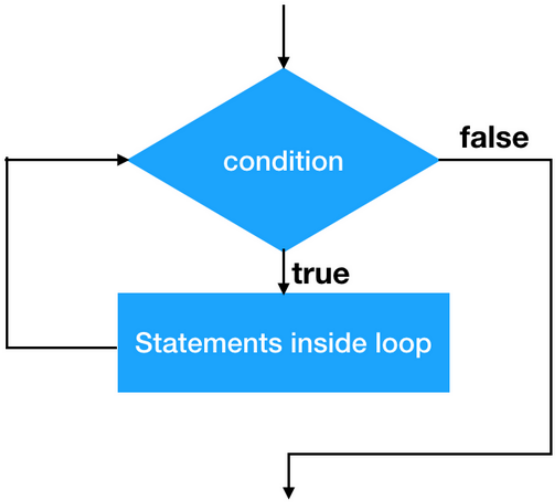

```c
#include <stdio.h> // prog150.c

int main() {
    double nota;

    printf("Digite sua nota: ");
    scanf("%lf", &nota);

    printf("Voce %s no exame.\n",
        nota >= 7 ? "passou" : "reprovou");

    return 0;
}
```

Estruturas de Repetição

Estruturas de repetição (loops)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

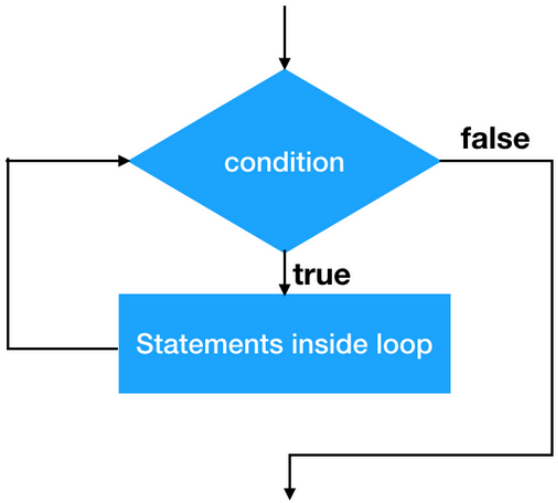
while loop




```
1 int contador = 0;
2
3 while (contador < 10) {
4   printf("%d ", contador);
5   contador++;
6 }
```

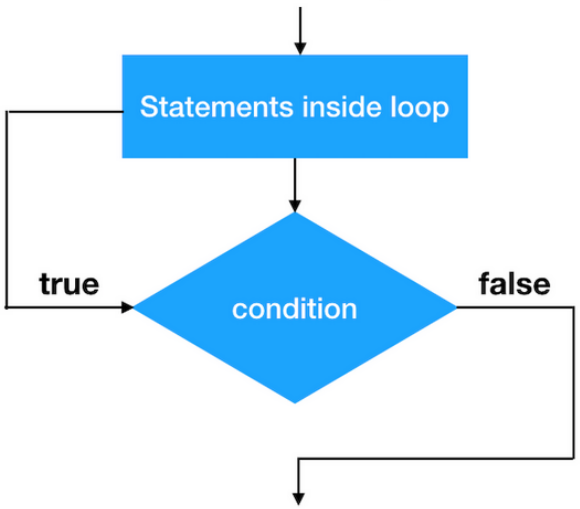
Estruturas de repetição (loops)

while loop




```c
int contador = 0;

while (contador < 10) {
  printf("%d ", contador);
  contador++;
}
```

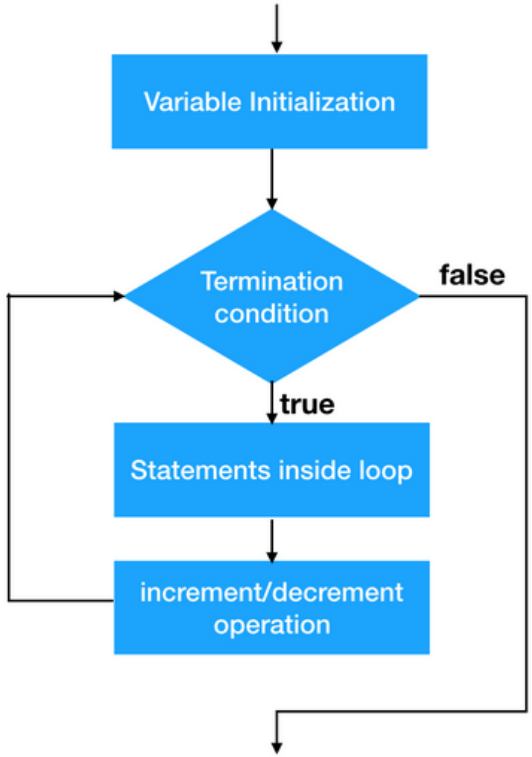
do..while loop




```c
int contador = 0;

do {
  printf("%d ", contador);
  contador++;
} while (contador < 10);
```

Estruturas de repetição (loops)

for loop




```
1 for (int i = 0; i < 10; i++) {
2     printf("%d ", i);
3 }
```

Vetores

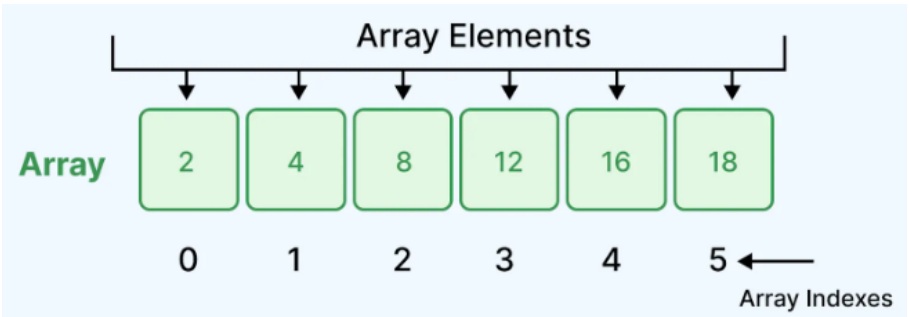
Vetores (Arrays)

- O C também suporta **variáveis agregadas**, que podem armazenar coleções de valores. Existem dois tipos de variáveis agregadas:
  - **arrays:** dados homogêneos
  - **structs:** permite dados heterogêneos

Vetores (Arrays)

- O C também suporta **variáveis agregadas**, que podem armazenar coleções de valores. Existem dois tipos de variáveis agregadas:
  - **arrays:** dados homogêneos
  - **structs:** permite dados heterogêneos

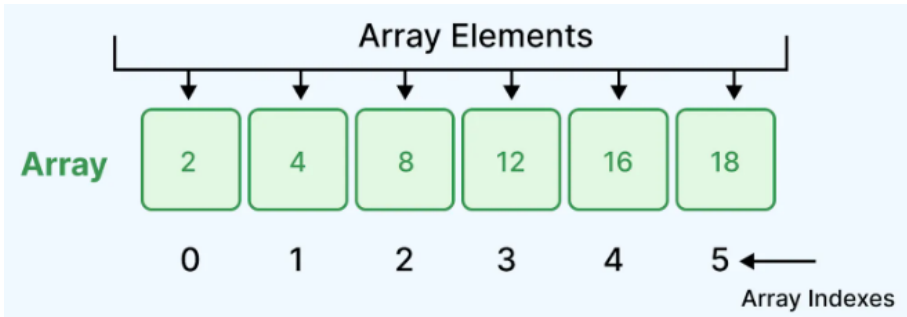
- Um vetor (array) é uma estrutura de dados que nos permite acessar muitas variáveis do mesmo tipo por meio de um único identificador.



Vetores (Arrays)

- O C também suporta **variáveis agregadas**, que podem armazenar coleções de valores. Existem dois tipos de variáveis agregadas:
  - **arrays:** dados homogêneos
  - **structs:** permite dados heterogêneos

- Um vetor (array) é uma estrutura de dados que nos permite acessar muitas variáveis do mesmo tipo por meio de um único identificador.



- Em quais casos na programação pode ser necessário o uso de um vetor?

Declarando e inicializando vetores

`int numbers[10];`	Um array com capacidade para 10 inteiros não inicializado.
`const int SIZE = 10;` `int numbers[SIZE];`	É uma boa ideia usar uma variável constante para o tamanho.
`int size = 10;` `int numbers[size];`	**Atenção:** Variable Length Arrays (VLA) desde C99.
`int vec[5] = {0,1,4,9,16};`	Um vetor de cinco inteiros, inicializado com "brace inicialization"
`int vec[] = {0,1,4,9,16};`	O tamanho do vetor pode ser omitido neste caso, pois ele é definido pelo número de valores iniciais.
`int squares[5] = {0,1,4};`	Se você fornecer menos valores iniciais que o tamanho, os valores restantes serão definidos como 0. Esse array contém 0, 1, 4, 0, 0.
`int squares[] = {[8] = 13};`	Inicializa somente os índices fornecidos.

Tamanho de um vetor

Se estivermos no mesmo escopo em que um array foi definido, podemos determinar seu tamanho usando o operador `sizeof`.

Exemplo:

Tamanho de um vetor

Se estivermos no mesmo escopo em que um array foi definido, podemos determinar seu tamanho usando o operador `sizeof`.

Exemplo:

```c
#include <stdio.h> // prog16.c

int main() {
  int array[] = { 1, 1, 2, 3, 5, 8, 13, 21 };

  size_t n = sizeof(array) / sizeof(array[0]);

  printf("Tamanho do array: %zu\n", n);

  return 0;
}
```

- **Obs.:** Só funcionará se `sizeof` estiver na mesma função/escopo no qual o vetor estiver declarado.

Exemplo de array


```c
#include <stdio.h> // prog17.c

int main() {
  int array[] = { 0, [1] = 1, [5] = 2, [8] = 13 };

  int n = sizeof(array) / sizeof(array[0]);

  for(int i = 0; i < n; ++i) {
    printf("[%d] = %d\n", i,  array[i]);
  }

  return 0;
}
```

Arrays multidimensionais (matrizes)

- Um array pode ter qualquer número de dimensões.
- A seguinte declaração cria uma matriz:

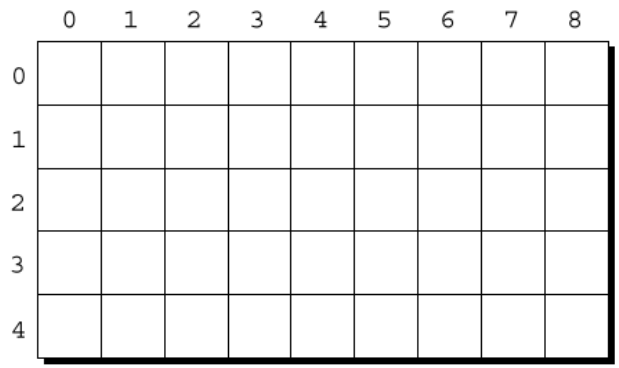
```
int mat[5][9];
```

Arrays multidimensionais (matrizes)

- Um array pode ter qualquer número de dimensões.
- A seguinte declaração cria uma matriz:

```
int mat[5][9];
```

  A matriz tem capacidade para 5 linhas e 9 colunas.

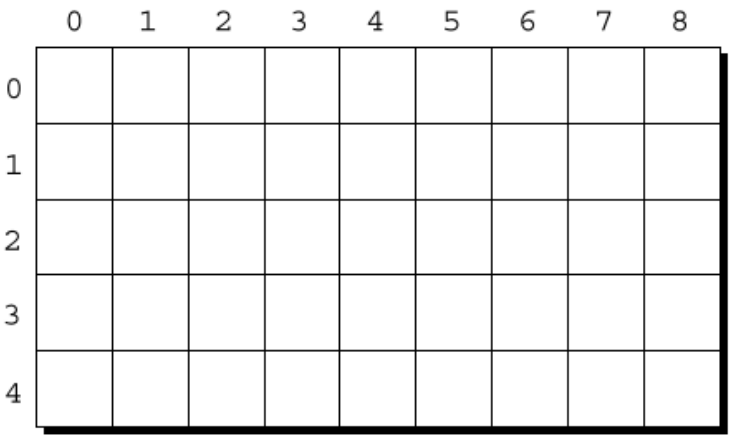


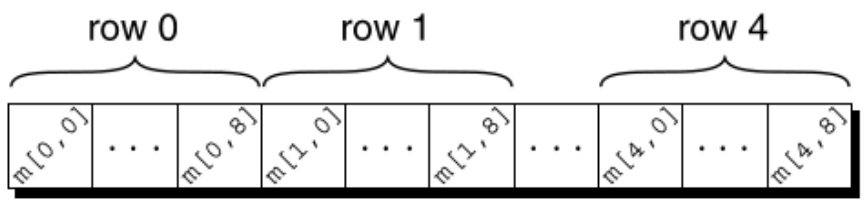
A fim de acessar o elemento na linha i e coluna j da matriz mat, devemos escrever $mat[i][j]$.

Representação interna de matrizes

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

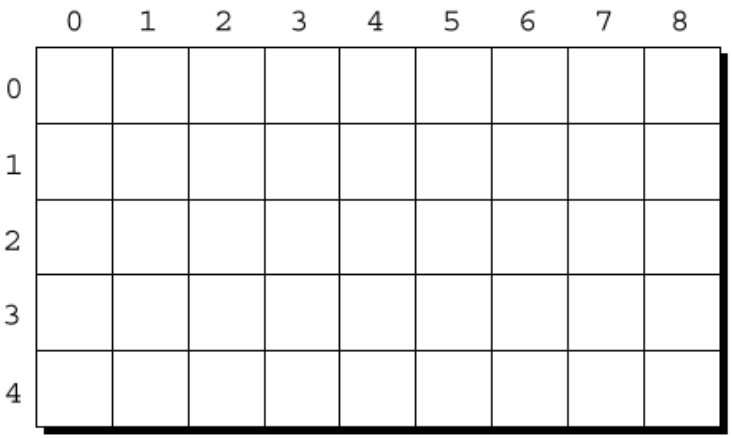
- Embora vizualizemos uma matriz como uma tabela, esse não é o jeito que os elementos estão de fato representados na memória.

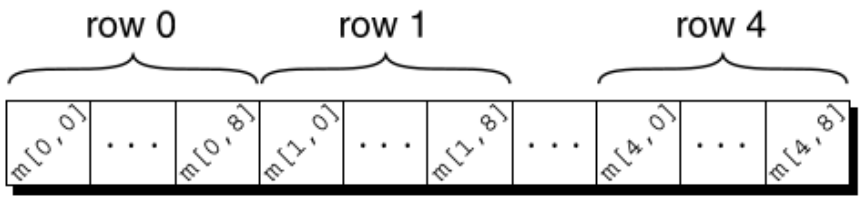




Representação interna de matrizes

- Embora vizualizemos uma matriz como uma tabela, esse não é o jeito que os elementos estão de fato representados na memória.
- C armazena matriz em uma ordem chamada **row-major order**, com a linha 0 primeiro, depois linha 1, e assim por diante.





Exemplo: Criando matriz identidade


```c
#include <stdio.h> // prog120.c

int main() {
    const int N = 5;
    int mat[N][N];

    for(int row = 0; row < N; ++row) {
        for(int col= 0; col < N; ++col) {
            if(row == col)
                mat[row][col] = 1;
            else
                mat[row][col] = 0;
        }
    }

    for(int row = 0; row < N; ++row) {
        for(int col= 0; col < N; ++col) {
            printf("%d ", mat[row][col]);
        }
        printf("\n");
    }
    return 0;
}
```

Arrays de caracteres – Strings do C

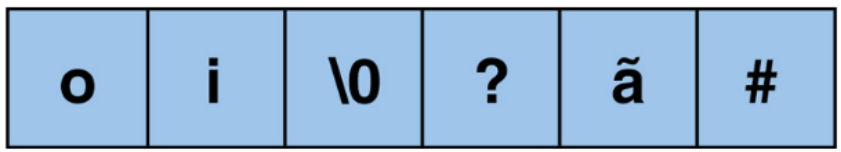
Arrays de caracteres — Strings do C

- Na linguagem C, uma string é uma sequência de caracteres terminados pelo caractere '\0'.

Arrays de caracteres — Strings do C

- Na linguagem C, uma string é uma sequência de caracteres terminados pelo caractere '\0'.
  - É armazenada na memória do computador na forma de um array do tipo char. Exemplo de declaração + inicialização:

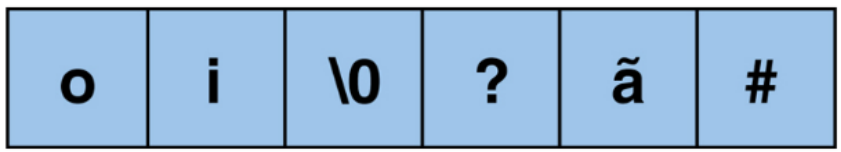
```
char palavra[6] = ''oi'';
```



Arrays de caracteres — Strings do C

- Na linguagem C, uma string é uma sequência de caracteres terminados pelo caractere '\0'.
  - É armazenada na memória do computador na forma de um array do tipo char. Exemplo de declaração + inicialização:

```
char palavra[6] = ''oi'';
```

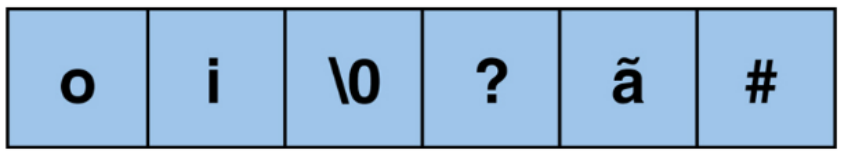


  - Nesse caso, temos 3 posições não utilizadas e que estão preenchidas com lixo de memória (um valor qualquer).

Arrays de caracteres — Strings do C

- Na linguagem C, uma string é uma sequência de caracteres terminados pelo caractere '\0'.
  - É armazenada na memória do computador na forma de um array do tipo char. Exemplo de declaração + inicialização:

```
char palavra[6] = ''oi'';
```



  - Nesse caso, temos 3 posições não utilizadas e que estão preenchidas com lixo de memória (um valor qualquer).

  - Como o caractere $\backslash 0$ indica o final de nossa string, isso significa que, em uma string definida com tamanho de 6 caracteres, apenas 5 estarão disponíveis para armazenar o texto digitado pelo usuário.

Strings do C — Inicialização

- Inicialização de strings: strings seguem as mesmas regras que os arrays: você pode inicializar a string no momento da criação, mas você não pode atribuir valores a ela usando o operador de atribuição depois disso.

Strings do C — Inicialização

- Inicialização de strings: strings seguem as mesmas regras que os arrays: você pode inicializar a string no momento da criação, mas você não pode atribuir valores a ela usando o operador de atribuição depois disso.

- Primeira forma de inicialização:

```
char palavra[6] = ‘‘oi’’;
```

Strings do C — Inicialização

- Inicialização de strings: strings seguem as mesmas regras que os arrays: você pode inicializar a string no momento da criação, mas você não pode atribuir valores a ela usando o operador de atribuição depois disso.

- Primeira forma de inicialização:

```
char palavra[6] = ‘‘oi’’;
```

- Segunda forma de inicialização:

```
char palavra[6] = { ‘o’, ‘i’, ‘\0’ };
```

Strings do C — Modificando uma string

- A linguagem C não suporta a atribuição de um array para outro. Logo, não é possível usar o operador de atribuição entre strings.

Strings do C — Modificando uma string

- A linguagem C não suporta a atribuição de um array para outro. Logo, não é possível usar o operador de atribuição entre strings.

```c
#include <stdio.h> // prog121.c

int main() {
    char palavra1[6] = "oi";
    char palavra2[6];

    palavra2 = palavra1; // ERRADO!!!

    return 0;
}
```

Strings do C — Modificando uma string

- A linguagem C não suporta a atribuição de um array para outro. Logo, não é possível usar o operador de atribuição entre strings.

```c
#include <stdio.h> // prog121.c

int main() {
    char palavra1[6] = "oi";
    char palavra2[6];

    palavra2 = palavra1; // ERRADO!!!

    return 0;
}
```

- Para atribuir o conteúdo de uma string a outra, o correto é copiar a string, elemento por elemento, para a outra string.

Exemplo: copiando uma string


```c
#include <stdio.h>

int main() {
    char str1[6] = "solar";
    char str2[6];
    int i = 0;

    for(i = 0; str1[i] != '\0'; ++i) {
        str2[i] = str1[i];
    }
    str2[i] = '\0';

    printf("%s\n", str2);
    return 0;
}
```

Observação

- Infelizmente, o tipo de manipulação de arrays mostrado no código do slide anterior não é muito prático quando estamos trabalhando com palavras.

Observação

- Infelizmente, o tipo de manipulação de arrays mostrado no código do slide anterior não é muito prático quando estamos trabalhando com palavras.
- Felizmente, a biblioteca-padrão da linguagem C possui funções especialmente desenvolvidas para a manipulação de strings na biblioteca <string.h>

Lendo strings do teclado — `fgets()`

- A função `fgets()` faz a leitura do teclado considerando todos os caracteres digitados (incluindo os espaços) até encontrar uma tecla enter:

```
char *fgets(char *str, int tamanho, FILE *fp);
```

Lendo strings do teclado — `fgets()`

- A função `fgets()` faz a leitura do teclado considerando todos os caracteres digitados (incluindo os espaços) até encontrar uma tecla enter:

```
char *fgets(char *str, int tamanho, FILE *fp);
```

- A função `fgets()` recebe três parâmetros de entrada:

Lendo strings do teclado — `fgets()`

- A função `fgets()` faz a leitura do teclado considerando todos os caracteres digitados (incluindo os espaços) até encontrar uma tecla enter:

```
char *fgets(char *str, int tamanho, FILE *fp);
```

- A função `fgets()` recebe três parâmetros de entrada:
  - **str**: a string a ser lida.

Lendo strings do teclado — `fgets()`

- A função `fgets()` faz a leitura do teclado considerando todos os caracteres digitados (incluindo os espaços) até encontrar uma tecla enter:

```
char *fgets(char *str, int tamanho, FILE *fp);
```

- A função `fgets()` recebe três parâmetros de entrada:
  - **str**: a string a ser lida.
  - **tamanho**: o limite máximo de caracteres a serem lidos, incluindo o '$\backslash 0$'.

Lendo strings do teclado — `fgets()`

- A função `fgets()` faz a leitura do teclado considerando todos os caracteres digitados (incluindo os espaços) até encontrar uma tecla enter:

```
char *fgets(char *str, int tamanho, FILE *fp);
```

- A função `fgets()` recebe três parâmetros de entrada:
  - **str**: a string a ser lida.
  - **tamanho**: o limite máximo de caracteres a serem lidos, incluindo o '$\backslash 0$'.
  - **fp**: a variável que está associada ao arquivo de onde a string será lida.

Lendo strings do teclado — `fgets()`

- A função `fgets()` faz a leitura do teclado considerando todos os caracteres digitados (incluindo os espaços) até encontrar uma tecla enter:

```
char *fgets(char *str, int tamanho, FILE *fp);
```

- A função `fgets()` recebe três parâmetros de entrada:
  - **str**: a string a ser lida.
  - **tamanho**: o limite máximo de caracteres a serem lidos, incluindo o '$\backslash 0$'.
  - **fp**: a variável que está associada ao arquivo de onde a string será lida.

- Ela retorna um ponteiro para o primeiro caractere da string recuperada em `str`, ou NULL caso dê algum erro.

Lendo strings do teclado — `fgets()`

- A função `fgets()` faz a leitura do teclado considerando todos os caracteres digitados (incluindo os espaços) até encontrar uma tecla enter:

```
char *fgets(char *str, int tamanho, FILE *fp);
```

- A função `fgets()` recebe três parâmetros de entrada:
  - **str**: a string a ser lida.
  - **tamanho**: o limite máximo de caracteres a serem lidos, incluindo o '$\backslash 0$'.
  - **fp**: a variável que está associada ao arquivo de onde a string será lida.

- Ela retorna um ponteiro para o primeiro caractere da string recuperada em `str`, ou NULL caso dê algum erro.

- **Atenção:** Na função `fgets()`, o caractere de nova linha ('$\backslash n$') fará parte da string se ele couber no array. Sempre adiciona o terminador '$\backslash 0$' ao final da string lida.

Exemplo de uso — `fgets()`


```c
#include <stdio.h>

int main(void) {
    char buffer[20];

    printf("Digite algo: ");
    if (fgets(buffer, sizeof(buffer), stdin) != NULL) {
        // note que pode incluir '\n'
        printf("Você digitou: %s", buffer);
    } else {
        printf("Erro na leitura!\n");
    }

    return 0;
}
```

Limpando o buffer do teclado (`stdin`)

- Às vezes, podem ocorrer erros durante a leitura de caracteres ou strings do teclado e alguns caracteres podem sobrar no buffer de entrada `stdin`.

Limpando o buffer do teclado (`stdin`)

- Às vezes, podem ocorrer erros durante a leitura de caracteres ou strings do teclado e alguns caracteres podem sobrar no buffer de entrada `stdin`.

- Assim, dvemos limpar o buffer do teclado (entrada-padrão) antes de ler novas strings.

Limpando o buffer do teclado (`stdin`)

- Às vezes, podem ocorrer erros durante a leitura de caracteres ou strings do teclado e alguns caracteres podem sobrar no buffer de entrada `stdin`.

- Assim, dvemos limpar o buffer do teclado (entrada-padrão) antes de ler novas strings.

- Uma forma segura de fazer isso é utilizar a função `getchar()` para ler os caracteres restantes anter encontrar um caractere de nova linha ('$\backslash n$') ou a constante EOF.

Limpando o buffer do teclado (`stdin`)

- Às vezes, podem ocorrer erros durante a leitura de caracteres ou strings do teclado e alguns caracteres podem sobrar no buffer de entrada `stdin`.

- Assim, dvemos limpar o buffer do teclado (entrada-padrão) antes de ler novas strings.

- Uma forma segura de fazer isso é utilizar a função `getchar()` para ler os caracteres restantes anter encontrar um caractere de nova linha ('$\backslash n$') ou a constante EOF.

- Exemplo:

Limpando o buffer do teclado (stdin)

- Às vezes, podem ocorrer erros durante a leitura de caracteres ou strings do teclado e alguns caracteres podem sobrar no buffer de entrada `stdin`.

- Assim, dvemos limpar o buffer do teclado (entrada-padrão) antes de ler novas strings.

- Uma forma segura de fazer isso é utilizar a função `getchar()` para ler os caracteres restantes anter encontrar um caractere de nova linha ('$\backslash n$') ou a constante EOF.

- Exemplo:

```
1 void clear_stdin() {
2     int c;
3     // descarta todos os caracteres até '\n' ou EOF
4     while ((c = getchar()) != '\n' && c != EOF);
5 }
```

Exemplo: leitura + limpeza


```c
#include <stdio.h> // prog126.c
#include <string.h>

void clear_stdin(void) {
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
}
```

```c
int main() {
    char buffer[5];        // buffer pequeno para exemplo
    int max_iter = 3;      // número de leituras

    for (int i = 0; i < max_iter; i++) {
        printf("Digite uma palavra (max 4 chars): ");

        if (fgets(buffer, 5, stdin) == NULL) {
            return 0; // erro ou EOF
        }

        size_t len = strlen(buffer);

        // Remove '\n' se estiver presente
        if (len > 0 && buffer[len - 1] == '\n') {
            buffer[len - 1] = '\0';
        } else {
            // Sobrou excesso no stdin, limpar
            clear_stdin();
        }
        printf("Você digitou: %s\n", buffer);
    }
    return 0;
}
```

Escrevendo uma string na tela — `printf()`

- Basicamente, para escrever uma string na tela utilizamos a função `printf()` com o formato de dados "%s":

Escrevendo uma string na tela — `printf()`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Basicamente, para escrever uma string na tela utilizamos a função `printf()` com o formato de dados "%s":

- Exemplo:

```
char str[20] = ‘‘Hello World’’;
printf(‘‘%s’’,str);
```

Escrevendo uma string na tela — `fputs()`

- Outra função que também permite a escrita de strings na tela. Essa função é a `fputs()`, cujo protótipo é:

```
int fputs(char *str, FILE *fp);
```

Escrevendo uma string na tela — `fputs()`

- Outra função que também permite a escrita de strings na tela. Essa função é a `fputs()`, cujo protótipo é:

```
int fputs(char *str, FILE *fp);
```

- A função `fputs()` recebe dois parâmetros de entrada:

Escrevendo uma string na tela — `fputs()`

- Outra função que também permite a escrita de strings na tela. Essa função é a `fputs()`, cujo protótipo é:

```
int fputs(char *str, FILE *fp);
```

- A função `fputs()` recebe dois parâmetros de entrada:
  - **str**: a string (array de caracteres) a ser escrita na tela.

Escrevendo uma string na tela — `fputs()`

- Outra função que também permite a escrita de strings na tela. Essa função é a `fputs()`, cujo protótipo é:

```
int fputs(char *str, FILE *fp);
```

- A função `fputs()` recebe dois parâmetros de entrada:
  - **str**: a string (array de caracteres) a ser escrita na tela.
  - **fp**: a variável que está associada ao arquivo onde a string será escrita.

Escrevendo uma string na tela — `fputs()`

- Outra função que também permite a escrita de strings na tela. Essa função é a `fputs()`, cujo protótipo é:

```
int fputs(char *str, FILE *fp);
```

- A função `fputs()` recebe dois parâmetros de entrada:
  - **str**: a string (array de caracteres) a ser escrita na tela.
  - **fp**: a variável que está associada ao arquivo onde a string será escrita.

- `fputs()` retorna a constante EOF (em geral, –1), se houver erro na escrita, ou um valor diferente de ZERO, se o texto for escrito com sucesso.

Escrevendo uma string na tela — `fputs()`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h>

int main() {
    char str[15] = "paralelepipedo";

    fputs(str, stdout);
    fputc('\n', stdout); // imprime apenas um caractere

    return 0;
}
```

Manipulando strings (biblioteca `string.h`)

Duas funções da biblioteca `string.h`:

- `int strlen(const char *str);`
  Retorna o número de caracteres da string `str`, excluindo o caractere nulo de terminação.

Manipulando strings (biblioteca `string.h`)

Duas funções da biblioteca `string.h`:

- `int strlen(const char *str);`
  Retorna o número de caracteres da string `str`, excluindo o caractere nulo de terminação.

- `char* strcpy(char *destino, const char *origem);`
  Copia a string `origem` para a string `destino`, incluindo o caractere nulo de terminação (e parando nesse ponto) e retorna o endereço da string destino.
  Para evitar overflows, o tamanho do vetor `destino` deve ser longo o suficiente para conter a string `origem`.

Exemplo com strlen e strcpy

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog20.c
#include <string.h>

int main () {
   char str1[] = "oi gente";
   char str2[40];
   char str3[40];

   strcpy(str2,str1);

   strcpy(str3, "Copia bem sucedida");

   printf("str1: %s, str2: %s\n", str1, str2);
   printf("str3: %s\n", str3);
   printf("tamanho de str1: %zu\n", strlen(str1));

   return 0;
}
```

Comparando duas strings

- Da mesma maneira como o operador de atribuição não funciona para strings, o mesmo ocorre com operadores relacionais usados para comparar duas strings. Desse modo, para saber se duas strings são iguais usa-se a função `strcmp()`:

```
int strcmp(const char *str1, const char *str2);
```

Comparando duas strings

- Da mesma maneira como o operador de atribuição não funciona para strings, o mesmo ocorre com operadores relacionais usados para comparar duas strings. Desse modo, para saber se duas strings são iguais usa-se a função `strcmp()`:

```
int strcmp(const char *str1, const char *str2);
```

- A função `strcmp()` compara posição a posição as duas strings (str1 e str2) e retorna um valor inteiro igual a zero, no caso de as duas strings serem iguais. Um valor de retorno diferente de zero significa que as strings são diferentes.

Exemplo com a função strcmp()

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog200.c
#include <string.h>

int main() {
    char str1[30], str2[30];

    printf("digite a primeira string (max 29 chars): ");
    fgets(str1, 30, stdin);
    printf("digite a segunda string (max 29 chars):  ");
    fgets(str2, 30, stdin);

    if(strcmp(str1, str2) == 0) {
        printf("strings iguais\n");
    } else {
        printf("strings diferentes\n");
    }

    return 0;
}
```

Concatenando duas strings

- A **concatenação** consiste em copiar uma string para o final de outra string.

Concatenando duas strings

- A **concatenação** consiste em copiar uma string para o final de outra string.

- Na linguagem C, para fazer a concatenação de duas strings, usa-se a função `strcat()`:

```
char* strcat(char *destino, const char *origem);
```

Concatenando duas strings

- A **concatenação** consiste em copiar uma string para o final de outra string.

- Na linguagem C, para fazer a concatenação de duas strings, usa-se a função `strcat()`:

```
char* strcat(char *destino, const char *origem);
```

- Basicamente, a função `strcat()` copia a sequência de caracteres contida em origem para o final da string destino e retorna o endereço da string destino.

Concatenando duas strings

- A **concatenação** consiste em copiar uma string para o final de outra string.

- Na linguagem C, para fazer a concatenação de duas strings, usa-se a função `strcat()`:

```
char* strcat(char *destino, const char *origem);
```

- Basicamente, a função `strcat()` copia a sequência de caracteres contida em origem para o final da string destino e retorna o endereço da string destino.

- O primeiro caractere da string contida em origem é colocado no lugar do caractere '$\backslash 0$' da string destino.

Exemplo com a função strcat()


```c
#include <stdio.h> // prog201.c
#include <string.h>

int main() {
    char destino[30] = "bom ";
    char origem[30] = "dia";

    strcat(destino, origem);

    printf("%s\n", destino);

    return 0;
}
```

Biblioteca `ctype.h`

<ctype.h>: útil ao se trabalhar com caracteres.

int isalnum(int c)	Verifica se c é um dígito decimal ou uma letra maiúscula ou minúscula.
int isalpha(int c)	Verifica se c é uma letra alfabética.
int isblank(int c)	Verifica se c é um caractere em branco (espaço ou tab)
int isdigit(int c)	Verifica se c é um caractere de dígito decimal.
int islower(int c)	Verifica se c é uma letra minúscula.
int isupper(int c)	Verifica se o parâmetro c é uma letra alfabética maiúscula.
int isspace(int c)	Verifica se c é um caractere de espaço em branco, ou seja, ' ' (space), '\t' (horizontal tab), '\n' (newline), '\v' (vertical tab), '\f' (feed), '\r' (carriage return)

Exemplo — ctype.h


```c
#include <stdio.h> // prog347.c
#include <ctype.h>

int main() {
    char ch;
    printf("Digite um caractere: ");

    while((ch = getchar()) != EOF) {
        printf("o caractere \'%c\': ", ch);
        if(isalnum(ch)) printf("\né alfanumérico");
        else printf("\nnão é alfanumérico");
        if(isdigit(ch)) printf("\né um dígito decimal");
        else printf("\nnão é um dígito decimal");
        if(isspace(ch)) printf("\né um espaço");
        else printf("\nnão é um espaço");
        if(islower(ch)) printf("\neh minusculo");
        else printf("\nnão é minúsculo");
        if(isupper(ch)) printf("\né maiúsculo");
        else printf("\nnão é maiúsculo");
        printf("\nDigite um caractere: ");
        getchar(); // limpa buffer
    }
}
```

Convertendo strings para números

Conversão de string para tipos nativos

Função	**Tipo retornado**

`int atoi(const char *nptr);`
`long atol(const char *nptr);`
`long long atoll(const char *nptr);`
	`int`
	`long`
	`long long`

`double atof(const char *nptr);`	`double`

Exemplo: conversão de string para numérico

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog120a.c
#include <stdlib.h>

int main(void) {
    char b1[10] = "123";
    char b2[10] = "123456789";
    char b3[10] = "12.345";
    char b4[17] = "2.546312891234";

    int a = atoi(b1);
    long b = atol(b2);
    float c = (float) atof(b3);
    double d = atof(b4);

    printf("%d %ld %f %.12f\n", a, b, c, d);

    return 0;
}
```

Conversão de tipos nativos para string

- Na linguagem C, é possível converter de tipos nativos para char*
- Para isso, existe a função `snprintf()`:

```
int snprintf(char *str, size_t size, const char *format, ...);
```

- Escreve no máximo `size-1` caracteres em `str` (mais o '$\backslash 0$' no fim). Retorna o número de caracteres que teria escrito (sem contar o '$\backslash 0$').

Exemplo: conversão de numérico para string

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
#include <stdio.h> // prog120b.c

int main(void) {
    char buf[20];
    int a = 255;
    long t = 123456789;
    float b = 347.561987;
    double c = 569.78961234;

    snprintf(buf, sizeof(buf), "%d", a);
    printf("buf = %s\n", buf);

    snprintf(buf, sizeof(buf), "%ld", t);
    printf("buf = %s\n", buf);

    snprintf(buf, sizeof(buf), "%.2f", b);
    printf("buf = %s\n", buf);

    snprintf(buf, sizeof(buf), "%.8f", c);
    printf("buf = %s\n", buf);

    return 0;
}
```

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM