

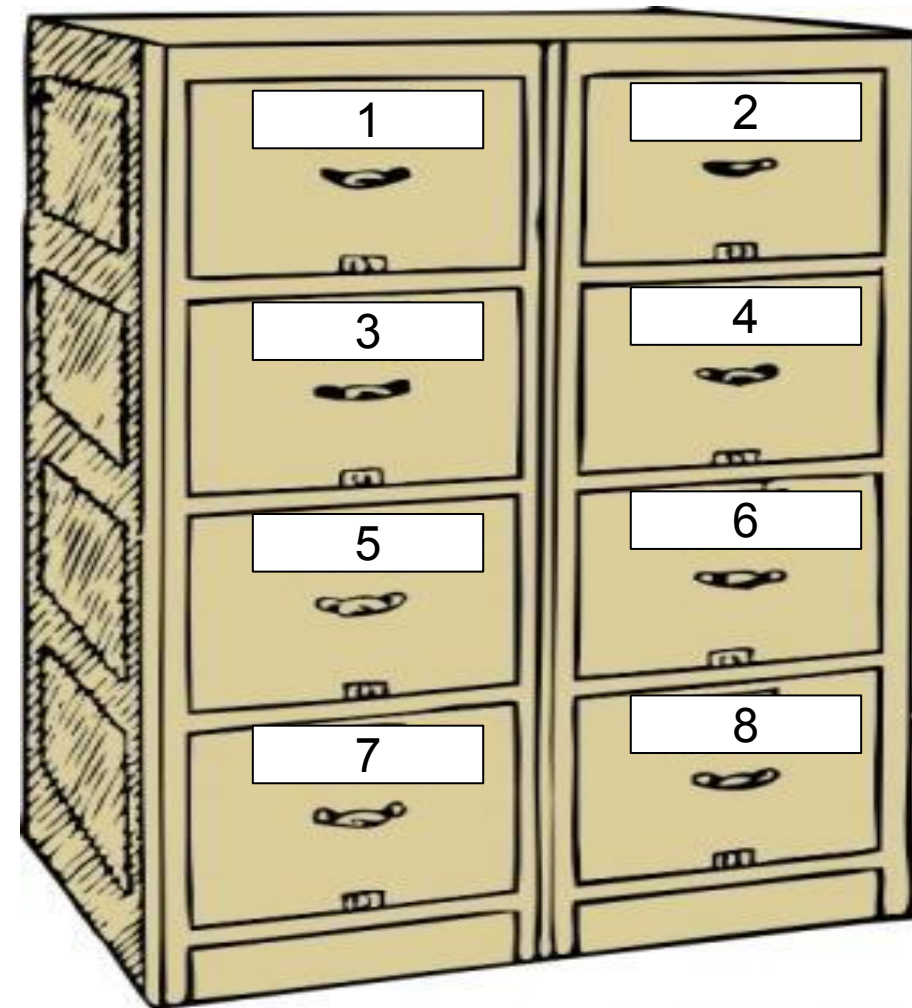
Programação e Desenvolvimento de Software I

Variáveis indexadas

Prof. Héctor Azpúrua
(slides adaptados do Prof. Pedro Olmo)

Variáveis indexadas

- **Por que índices são importantes?**
 - Como uma loja de sapatos artesanais deve guardar os seus produtos?
 - Tamanhos entre 35 e 42
 - Um único par por tamanho
 - Produção em ordem aleatória:
 - Ex: 39, 35, 42, 41, 37, 36, 38, 40
 - Problema:
 - Se são inseridos na ordem em que foram produzidos, vou ter que procurar todas as caixas...
 - **Solução “naive”:** $Gaveta = tamanho - 34$



Armário da loja

Variáveis indexadas

O que são?

■ Variável:

- Entidade que permite o acesso a uma posição de memória
- Usa um tipo de dado específico
- Faz referência a um nome simbólico
 - Exemplo: `contador`, `num_cachorros`

■ Variável Indexada:

- Conjunto de variáveis do mesmo tipo, referenciadas pelo mesmo nome e **individualizadas por índices**
- Podem ter um ou mais índices:
 - **Dimensão** é o número de índices necessários para localizar um elemento
- **Vetor** é uma variável indexada com uma dimensão
- **Matriz** é uma variável indexada com duas dimensões

Representação de variáveis indexadas

- Imagine a declaração da variável produtos como:

```
int n = 50;
```

```
float produtos[n];
```

- Se n é uma constante inteira, então os símbolos $[n]$ após o nome da variável indicam que ela poderá ocupar até **n posições de memória consecutivas**
- Logo, a variável texto poderá ocupar até 50 posições do tipo **float**
- Como as posições de memória são **consecutivas**, cada uma delas pode ser identificada por um **índice**
 - Igual nossa **string**!

Problema I

- Uma loja pretende **simular um dia de suas vendas**:
 - Sabe-se que os preços dos produtos vendidos nesta loja variam de R\$5,00 a R\$100,00 e que cada cliente compra apenas um produto
- O número e o preço de cada produto, o número de clientes e os produtos comprados pelos clientes devem ser gerados aleatoriamente:
 - No máximo 200 produtos
 - No máximo 50 clientes
- **Como podemos fazer?**


Problema I

- Precisamos de uma lista de produtos (máximo 200) e clientes (máximo 50)
 - Criados de forma **manual!**

Preços dos produtos

Id	Produto	Valor
0		
1		
2		
3		
4		
...		

Constante utilizando macros!



```
#define MAX_PRODUTOS 200

int main() {
    float precos[MAX_PRODUTOS];
    precos[0] = 5.99;
    precos[1] = 7.04;
    precos[2] = 99.99;
    precos[3] = 1.99;
    precos[4] = 42.50;
    return 0;
}
```

Problema I

- Uma loja pretende **simular um dia de suas vendas**:
 - Sabe-se que os preços dos produtos vendidos nesta loja variam de R\$5,00 a R\$100,00 e **que cada cliente compra apenas um produto**
- O número e o preço de cada produto, o número de clientes e os produtos comprados pelos clientes devem ser gerados aleatoriamente:
 - No máximo 200 produtos
 - No máximo 50 clientes
- **Como podemos fazer?**

Problema I

- Precisamos de uma lista de produtos (máximo 200) e clientes (máximo 50)
 - Criados de forma **manual!**

Preços dos produtos

Id Produto	Valor
0	5.99
1	7.04
2	99.99
3	1.99
4	42.50
...	...

Clientes

Id Cliente
0
1
2
3
4
...

Problema I

- Precisamos de uma lista de produtos (máximo 200) e clientes (máximo 50)

```
#define MAX_PRODUTOS 200
#define MAX_CLIENTES 50

int main() {
    float precos[MAX_PRODUTOS];
    int compras[MAX_CLIENTES];
    return 0;
}
```

- **Problema:**

- Como podemos **preencher** (popular) os nossos arranjos de preços e clientes?

Problema I

- Populando o arranjo de preços dos produtos...

```
#define MAX_PRODUTOS 200  
#define MAX_CLIENTES 50
```

```
int main() {  
    float precos[MAX_PRODUTOS];  
    for (int i = 0; i < MAX_PRODUTOS; i++) {  
        precos[i] = 1.99;  
    }  
}
```

```
    int compras[MAX_CLIENTES];  
    return 0;  
}
```

Problema I

- Relacionando um cliente com uma venda
 - Lembrando: Um cliente só compra **um produto** por vez

```
#define MAX_PRODUTOS 200  
#define MAX_CLIENTES 50
```

```
int main() {  
    float precos[MAX_PRODUTOS];  
    for (int i = 0; i < MAX_PRODUTOS; i++) {  
        precos[i] = 1.99;  
    }
```

```
int compras[MAX_CLIENTES];  
for (int i = 0; i < MAX_CLIENTES; i++) {  
    compras[i] = 32;  
}  
return 0;
```

```
}
```

Problema I

- Como calcular o valor total das compras feitas em um dia?

Preços dos produtos

Id Produto	Valor
0	1.99
1	1.99
2	1.99
3	1.99
4	1.99
...	...

Clientes

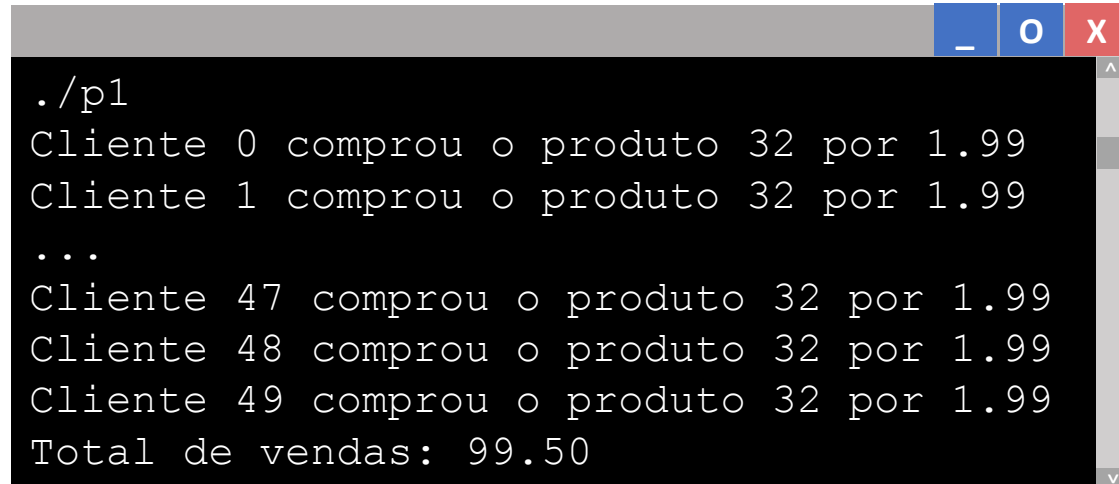
Id Cliente	Id Produto
0	32
1	32
2	32
3	32
4	32
...	...

Problema I

- Como calcular o valor total das compras feitas em um dia?

```
float total_vendas = 0.0;
for (int id_cliente = 0; id_cliente < MAX_CLIENTES; id_cliente++) {
    int id_produto = compras[id_cliente];
    float valor = precos[id_produto];
    total_vendas += valor;
    printf("Cliente %d comprou o produto %d por %.2f\n",
        id_cliente,
        id_produto,
        valor);
}
printf("Total de vendas: %.2f\n", total_vendas);
```

- Esse código tem um problema?
 - Todos os produtos e valores são iguais!

A terminal window with a dark background and white text. The window has a title bar with a minus sign, a maximize button, and a close button. The output shows a list of purchases where every client bought product 32 for 1.99, and a final total of 99.50.

```
./p1
Cliente 0 comprou o produto 32 por 1.99
Cliente 1 comprou o produto 32 por 1.99
...
Cliente 47 comprou o produto 32 por 1.99
Cliente 48 comprou o produto 32 por 1.99
Cliente 49 comprou o produto 32 por 1.99
Total de vendas: 99.50
```

Problema I

- Uma loja pretende **simular um dia de suas vendas**:
 - Sabe-se que os preços dos produtos vendidos nesta loja variam de R\$5,00 a R\$100,00 e que cada cliente compra apenas um produto
- O número e o preço de cada produto, o número de clientes e os produtos comprados pelos clientes devem ser **gerados aleatoriamente**:
 - No máximo 200 produtos
 - No máximo 50 clientes
- **Como podemos fazer?**

Problema I

- Precisamos de uma lista de produtos (máximo 200) e clientes (máximo 50)
 - Criados de forma **aleatoria!**

Preços dos produtos

Id	Produto	Valor
0		
1		
2		
3		
4		
...		

Clientes

Id	Cliente	Id	Produto
0			
1			
2			
3			
4			
...			

Geração de números aleatórios

Função rand

- A linguagem C dispõe da função **rand** para a geração de um inteiro aleatório no intervalo de 0 até `RAND_MAX`
 - `RAND_MAX` é uma constante definida em `stdlib.h`
- Especificamente:
 - `RAND_MAX = 0x7FFF = 32767`
- Para gerar números aleatórios em um intervalo específico podemos escrever uma função:

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

```
int random(int n) {
```

```
    return rand() % n;
```

```
}
```

Que valor é este aqui?

Um inteiro no intervalo $[0, n-1]$

Geração de números aleatórios

Função rand

- Sendo assim, para a geração do número de produtos, podemos utilizar:

```
num_produtos = 1 + random(MAX_PRODUTOS);
```

- Se MAX_PRODUTOS é uma constante que vale 200, tem-se:

- `num_produtos` $\in [1, 200]$

- Para a geração dos preços dos produtos, podemos escrever:

```
#include <stdlib.h>
int random(int n) {
    return rand() % n;
}
```

```
float preco = 5 + random(96);
precos[0] = preco;
```

preço $\in [5, 100]$

Geração de números aleatórios

Função rand

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

```
int randomInt(int n) {  
    return rand() % n;  
}
```

```
int main() {  
    for (int i = 0; i < 10; i++) {  
        int n = randomInt(20);  
        printf("%d ", n);  
    }  
    printf("\n");  
    return 0;  
}
```

```
./rand1  
7 9 13 18 10 12 4 18 3 9
```

São iguais!

Geração de números aleatórios

Sementes

- Ao trabalhar com números aleatórios, busca-se em cada execução do programa, **gerar novos números**
 - Para isso, ao se usar a função `rand`, é necessário fornecer uma **semente** à mesma... Como assim?
 - Os números gerados são **pseudo-aleatorios**
- Para uma dada semente, a função `rand` fará brotar uma determinada sequência de números
 - A sequencia sempre será a mesma para a mesma semente...
 - Para garantir que a sequência gerada seja sempre diferente, deve-se mudar a semente a cada execução!

Geração de números aleatórios

Sementes

- Como podemos mudar a semente?
 - Uma boa ideia é usar o valor retornado pela função `time`
 - Definida em `time.h`
 - A função `time` retorna como valor o número de segundos transcorridos desde 01/01/1970 e pode armazenar este valor num parâmetro

`time_t time(time_t *second)`

```
void main() {  
    time_t seconds;  
    // Stores time seconds  
    time(&seconds);  
    printf("%ld\n", seconds);  
}
```

```
void main() {  
    time_t seconds;  
    seconds = time(NULL);  
    printf("%ld\n", seconds);  
}
```

Geração de números aleatórios

Função `srand`

- A função `srand` é responsável pela inicialização da semente a ser usada pela função `rand`
- A função `srand` exige como parâmetro um valor inteiro do tipo `unsigned`
- Podemos escrever:

```
srand((unsigned)time(NULL));
```

Geração de números aleatórios

Função rand com srand

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

```
int randomInt(int n) {  
    return rand() % n;  
}
```

```
int main() {  
    srand((unsigned) time(NULL));  
    for (int i = 0; i < 10; i++) {  
        int n = randomInt(20);  
        printf("%d ", n);  
    }  
    printf("\n");  
    return 0;  
}
```

```
./srand  
2 0 6 2 7 7 14 15 10 2
```



Problema I

- Uma loja pretende **simular um dia de suas vendas**:
 - Sabe-se que os preços dos produtos vendidos nesta loja variam de R\$5,00 a R\$100,00 e que cada cliente compra apenas um produto
- O número e o preço de cada produto, o número de clientes e os produtos comprados pelos clientes devem ser **gerados aleatoriamente**:
 - No máximo 200 produtos
 - No máximo 50 clientes
- **Como podemos fazer?**

Problema I

- Agora criando os preços e associando clientes e produtos de **forma aleatória**

```
#define MIN_PRECO 5
#define MAX_PRECO 100
#define MAX_PRODUTOS 200
#define MAX_CLIENTES 5
```

```
int randomInt(int n) {
    return rand() % n;
}
```

```
int main() {
    srand((unsigned)time(NULL));

    float precos[MAX_PRODUTOS];
    for (int i = 0; i < MAX_PRODUTOS; i++) {
        precos[i] = MIN_PRECO + randomInt(MAX_PRECO);
    }
}
```

```
int compras[MAX_CLIENTES];
for (int i = 0; i < MAX_CLIENTES; i++) {
    compras[i] = randomInt(MAX_PRODUTOS);
}
```

```
float total_vendas = 0.0;
for (int id_cliente = 0;
     id_cliente < MAX_CLIENTES; id_cliente++) {
```

```
    int id_produto = compras[id_cliente];
    float valor = precos[id_produto];
    total_vendas += valor;
    printf("Cliente %d comprou %d por %.2f\n",
           id_cliente, id_produto, valor);
}
```

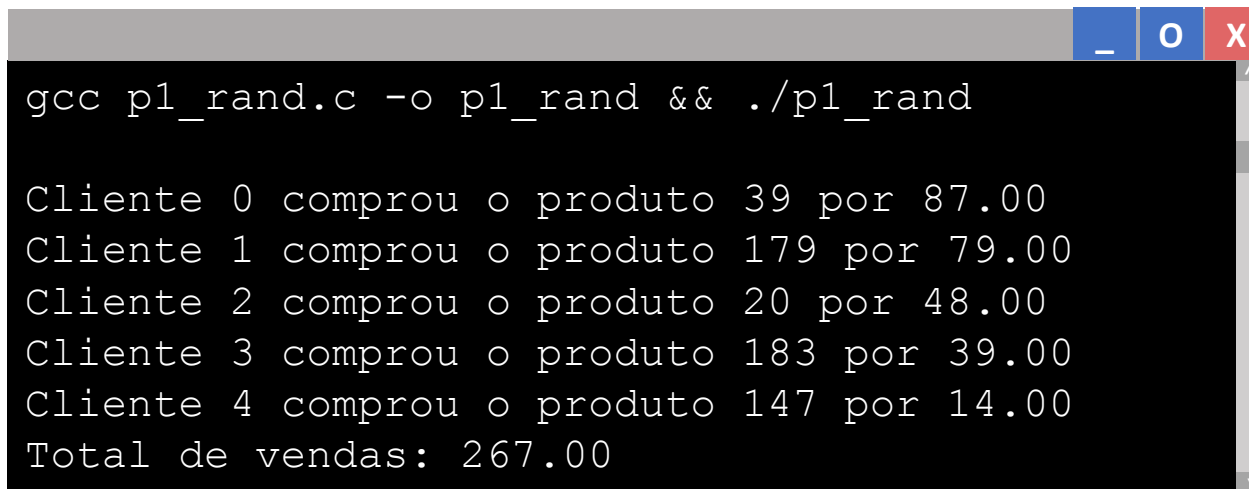
```
printf("Total de vendas: %.2f\n", total_vendas);
```

```
return 0;
```

```
}
```


Problema I

- Executando o programa anterior temos como resultado...

A terminal window with a dark background and white text. The window has a title bar with a minus sign, a maximize button, and a close button. The text inside the terminal shows the compilation and execution of a C program. The output lists purchases for five clients and a total sales amount.

```
gcc p1_rand.c -o p1_rand && ./p1_rand

Cliente 0 comprou o produto 39 por 87.00
Cliente 1 comprou o produto 179 por 79.00
Cliente 2 comprou o produto 20 por 48.00
Cliente 3 comprou o produto 183 por 39.00
Cliente 4 comprou o produto 147 por 14.00
Total de vendas: 267.00
```

- Para esta simulação, observa-se 5 clientes (0 a 4)
- Os produtos comprados por estes clientes estão armazenados na variável indexada **compras**

Variáveis indexadas

- Compras:

68	45	90	45	44			
0	1	2	3	4	5	...	49

- Exemplo: O cliente 0 comprou o produto 68

- Os preços estão armazenados na variável **preços**:

?	...	5	57	...	74	...	12	...	33	100
0	...	44	45	...	68	...	90	...	198	199

- Exemplo: O produto 68 tem preço igual a 74 reais e foi comprado pelo cliente 0

Variáveis indexadas

Índices

- Observe que o **valor de uma variável indexada** pode ser o **índice** de uma outra variável indexada
- Veja, por exemplo, o caso do cliente **0**:

`compras[0] = 68;`

`precos[68] = precos[compras[0]] = 73;`

- Ou seja, o **índice** de uma variável indexada precisa **ser um inteiro** com valor limitado ao número de posições de memória declaradas para a variável

Variáveis indexadas

Índices

- O **valor deste inteiro** pode ser o valor de uma **constante**, o valor de uma **expressão**, o valor de uma **variável** ou o valor retornado por uma **função**
- Assim, são válidas as expressões:

```
int x[10], i = 5, a;  
a = x[3];  
a = x[i];  
a = x[x[a]];  
a = x[random(10)];
```

Variáveis indexadas

Índices

■ **Atenção!**

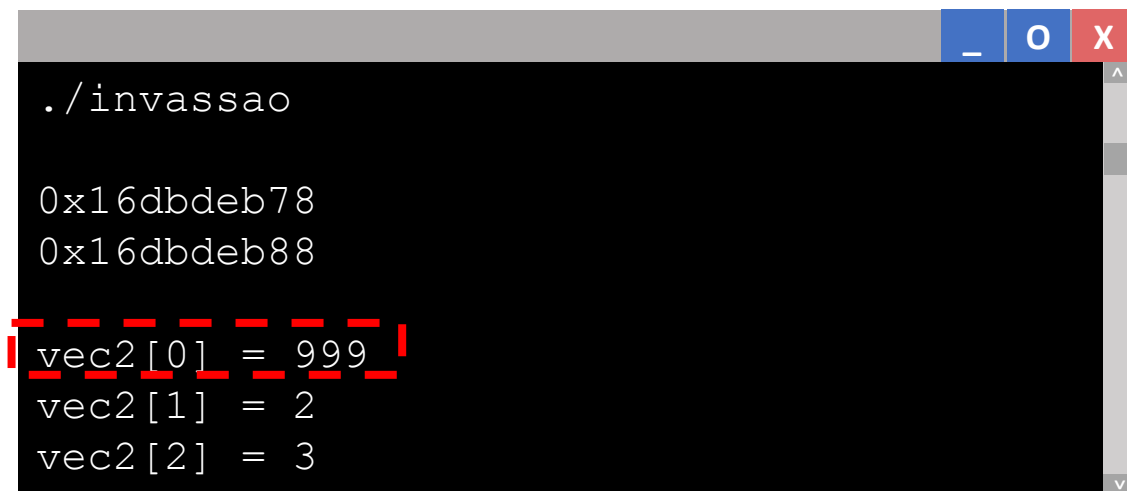
- Ao utilizar variáveis indexadas, temos que controlar o valor do índice no intervalo **0** a **$n-1$** , onde **n** é o número de posições de memória
- Caso contrário, podemos ter **invasão de memória**

Variáveis indexadas

Invasão de memória

Qual é o problema?

```
int main() {  
    int vec2[] = {1, 2, 3};  
    int vec1[] = {10, 20, 30};  
  
    printf("%p\n", vec1);  
    printf("%p\n", vec2);  
  
    for (int i = 0; i < 5; i++) {  
        vec1[i] = 999;  
    }  
  
    for (int i = 0; i < 3; i++) {  
        printf("vec2[%d] = %d\n", i, vec2[i]);  
    }  
    return 0;  
}
```



```
./invassao  
  
0x16dbdeb78  
0x16dbdeb88  
  
vec2[0] = 999  
vec2[1] = 2  
vec2[2] = 3
```

Variáveis indexadas

Invasão de memória

- O que esta acontecendo?
 - Ao escrever numa posição fora da memória alocada para a variável
 - Estamos acessando a memória de alguém mais!

```
for (int i = 0; i < 5; i++) {  
    vec1[i] = 999;  
}
```

- `vec1[0] = 999;`
- `vec1[1] = 999;`
- `vec1[2] = 999;`
- `vec1[3] = 999;`

Endereço	Variável	Valor
0x1	vec1[0]	999
0x2	vec1[1]	999
0x3	vec1[2]	999
0x4	vec2[0]	999
0x5	vec2[1]	2
0x6	vec2[2]	3
0x7		
0x8	vec1	0x1
0x9	vec2	0x4
0xA		

Problema 2

- Uma grande empresa armazena em uma variável indexada os números dos cheques emitidos num dia pelo setor financeiro
- Considere que os números dos cheques são valores inteiros de **1** a **100** e que os cheques são emitidos em uma ordem aleatória
- Ao final do dia, para facilitar o controle, a empresa precisa **ordenar** estes dados em **ordem crescente**

Ordenação por contagem

- A ordenação de dados é uma aplicação importante em computação, existindo vários algoritmos eficientes de ordenação
- O algoritmo que veremos para resolver este problema é um dos mais simples e menos eficientes: **ordenação por contagem**
- **Exemplo:** imagine uma variável indexada `vet` contendo os seguintes valores:

vet =	38	97	19	100	23	47	41	8
	0	1	2	3	4	5	6	7

Ordenação por contagem

- Considere ainda uma outra variável `pos`, de mesmo tamanho, com valores inicialmente iguais a zero:

`pos` =

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7

- O propósito da variável `pos[i]` é armazenar a posição que o elemento `vet[i]` deve ter na ordenação
- Para determinar esta posição, basta contar para cada elemento `vet[i]`, quantos elementos menores do que ele existem na variável `vet` e armazenar em `pos[i]`

Ordenação por contagem

Exemplo

vet =

38	97	19	100	23	47	41	8
0	1	2	3	4	5	6	7

i	vet[i]	Elementos menores do que vet[i]	pos[i]
0	38	19, 23, 8	3
1	97	38, 19, 23, 47, 41, 8	6
2	19	8	1
3	102	38, 97, 19, 23, 47, 41, 8	7
4	23	19, 8	2
5	47	38, 19, 23, 41, 8	5
6	41	38, 19, 23, 8	4
7	8	—	0

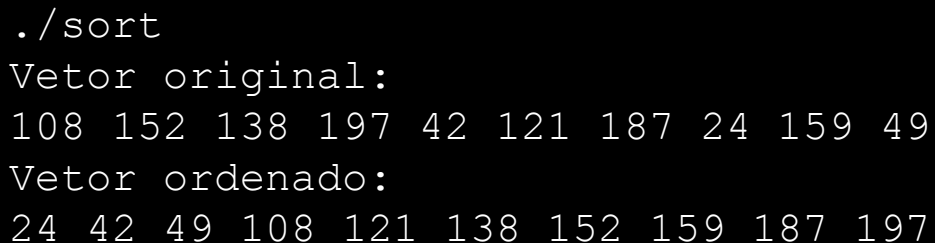
Ordenação por contagem

Exemplo

```
int randomInt(int n) {
    return rand() % n;
}

int existe(int *vec, int size, int num) {
    for (int i = 0; i < size; i++) {
        if (vec[i] == num)
            return 1;
    }
    return 0;
}

int contamenores(int *vec, int size, int num) {
    int count = 0;
    for (int i = 0; i < size; i++) {
        if (vec[i] < num)
            count++;
    }
    return count;
}
```



```
./sort
Vetor original:
108 152 138 197 42 121 187 24 159 49
Vetor ordenado:
24 42 49 108 121 138 152 159 187 197
```

```
int main() {
    int n = 10;
    srand((unsigned)time(NULL));
    int vec[n], pos[n], vec_ordenado[n];

    int i = 0;
    do {
        int rand_v = randomInt(200);
        if (!existe(vec, n, rand_v)) {
            vec[i] = rand_v;
            i++;
        }
    } while (i < n);

    for (int i = 0; i < n; i++) {
        pos[i] = contamenores(vec, n, vec[i]);
    }

    for (int i = 0; i < n; i++) {
        vec_ordenado[pos[i]] = vec[i];
    }

    printf("Vetor ordenado:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", vec_ordenado[i]);
    }
}
```

Vetores e matrizes

- Uma variável indexada pode ter uma ou mais **dimensões**
 - Considere, por exemplo, as declarações:

```
int a[10];  
char b[3][5];  
double c[2][2][3];
```
 - Dizemos que *a* tem **uma dimensão**, *b* tem **duas dimensões** e *c* tem **três dimensões**
 - Para as declarações anteriores, temos:

Variável	Nº de posições	Memória alocada (bytes)	Exemplo de referência
a	10	$10 * 4 = 40$	<code>a[i]</code>
b	$3 * 5 = 15$	$15 * 1 = 15$	<code>b[i][j]</code>
c	$2 * 2 * 3 = 12$	$12 * 8 = 96$	<code>c[i][j][k]</code>

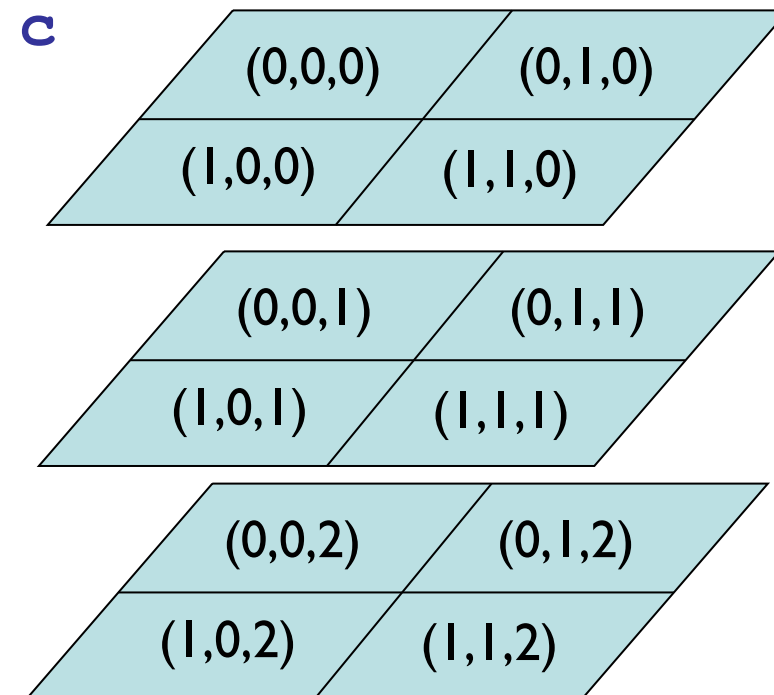
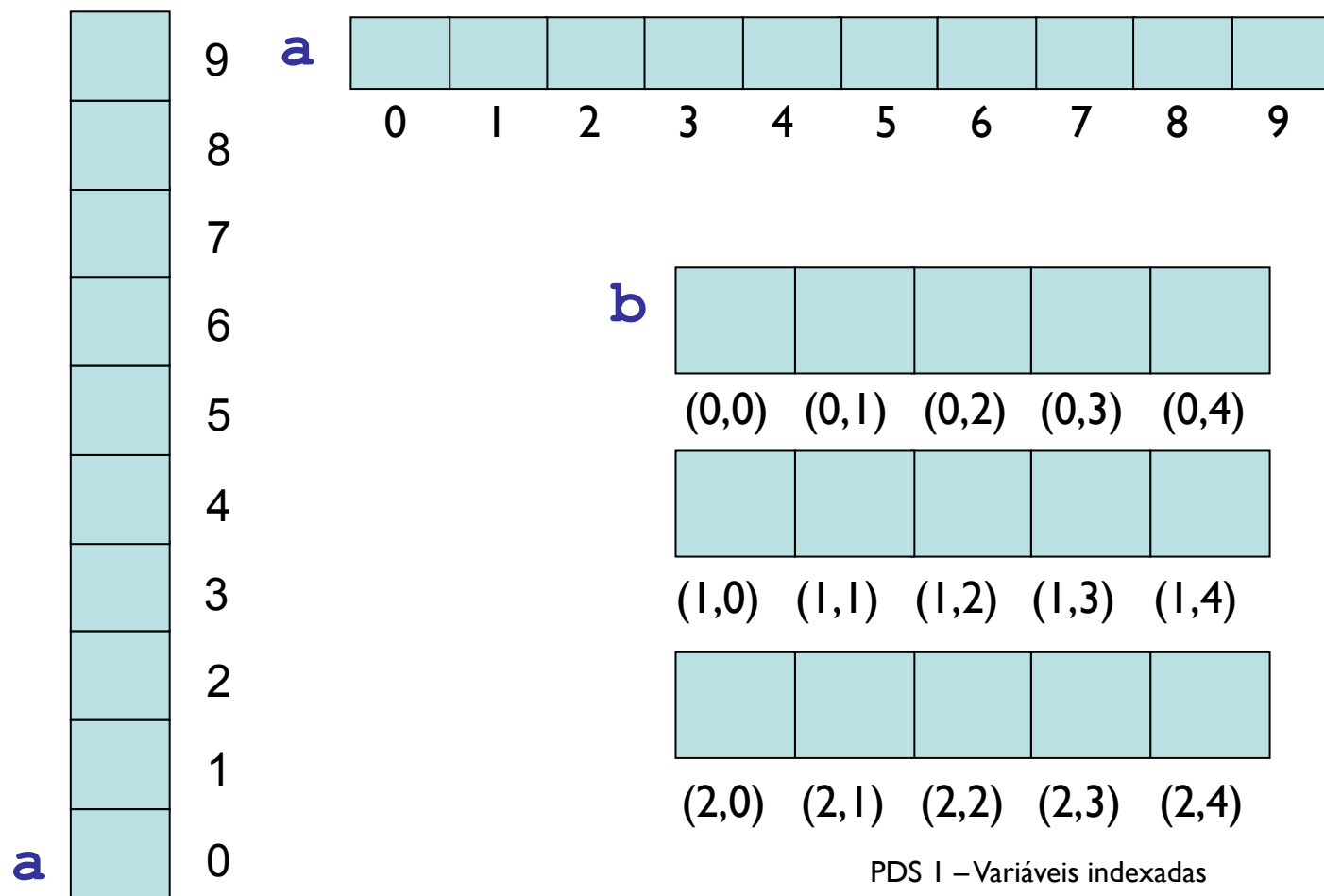
Vetores e matrizes

- Em programação, os termos **vetor** e **matriz** são usados como sinônimos de variável indexada
- O termo **vetor** é usado para uma variável indexada de uma única dimensão e o termo **matriz** para variáveis indexadas de duas ou mais dimensões
- Para uma melhor compreensão das variáveis indexadas, utiliza-se abstrações a respeito da disposição espacial de seus elementos

Vetores e matrizes

- Considere:

```
int a[10];  
char b[3][5];  
double c[2][2][3];
```




Problema 3

- Construa uma matriz para armazenar os resultados da simulação da rolagem de dois dados. O elemento $[i][j]$ da matriz armazena o número de vezes que o valor do primeiro dado é i e o valor do segundo dado é j
- Considere que um **vetor armazena a frequência de cada soma possível dos valores dos dados**
- Determinar qual é a soma mais frequente após rolar os dados 36.000 vezes
- **Imprima a matriz** que armazena os resultados das rolagens dos dados

Problema 3

- Pergunta:
 - Quais são as somas possíveis para os valores dos dados?
- Resposta:
 - [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] ou seja, existem **11 possibilidades**
 - Ou seja, precisamos de um vetor de 11 posições para armazenar as frequências das somas acima
- Exemplo:

3	125	23	41	5	9	2	86	231	45	13
0	1	2	3	4	5	6	7	8	9	10
										
2	3	4	5	6	7	8	9	10	11	12

Problema 3

Análise do programa

```
int main() {
    int d1, d2;
    int mat[6][6];
    int res[11];

    srand((unsigned)time(NULL));

    for (int i = 0; i < 36000; i++) {
        d1 = rand() % 6;
        d2 = rand() % 6;
        mat[d1][d2]++;
        res[d1 + d2]++;
    }

    printf("Matriz:\n");
    for (int i = 0; i < 6; i++) {
        for (int j = 0; j < 6; j++) {
            printf("%5d ", mat[i][j]);
        }
        printf("\n");
    }
}
```

```
printf("Resultados:\n");
for (int i = 0; i < 11; i++) {
    printf("%5d", res[i]);
}
printf("\n");

for (int i = 0; i < 11; i++) {
    printf("%5d", i + 2);
}
printf("\n");

int val = 0;
int rmf = -1;
for (int i = 0; i < 11; i++) {
    if (res[i] >= val) {
        val = res[i];
        rmf = i + 2;
    }
}
printf("Soma mais frequente: %d\n", rmf);
return 0;
}
```

Problema 3

Análise do programa

```
./mat1
Matriz:
  970   980  1016  1013   903   956
  977   983  1002  1000  1054  1012
 1029  1069   998   967  1004  1029
 1004   997  1020   952   995  1024
  984   998   962  1043   958   984
 1033   986  1049  1046  1010   993
Resultados:
  970 1957 3028 4088 4882 6028 4916 4116 3028 1994  993
    2    3    4    5    6    7    8    9   10   11   12
Soma mais frequente: 7
```

Problema 3

Análise do programa

```
int main() {
    int d1, d2;
    int mat[6][6];
    int res[11];

    srand((unsigned)time(NULL));

    for (int i = 0; i < 36000; i++) {
        d1 = rand() % 6;
        d2 = rand() % 6;
        mat[d1][d2]++;
        res[d1 + d2]++;
    }

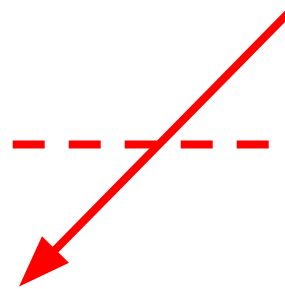
    printf("Matriz:\n");
    for (int i = 0; i < 6; i++) {
        for (int j = 0; j < 6; j++) {
            printf("%5d ", mat[i][j]);
        }
        printf("\n");
    }
}
```

```
printf("Resultados:\n");
for (int i = 0; i < 11; i++) {
    printf("%5d", res[i]);
}
printf("\n");
```

```
for (int i = 0; i < 11; i++) {
    printf("%5d", i + 2);
}
printf("\n");
```

```
int val = 0;
int rmf = -1;
for (int i = 0; i < 11; i++) {
    if (res[i] >= val) {
        val = res[i];
        rmf = i + 2;
    }
}
printf("Soma mais frequente: %d\n", rmf);
return 0;
```

Por que $i + 2$?



Problema 3

Exemplo de matrizes

- O que esta acontecendo?

```
if (res[i] >= val) {  
    val = res[i];  
    rmf = i + 2;  
}
```

- Lembre-se que o valor da soma corresponde ao **valor do índice do vetor + 2**

3	125	23	41	5	9	2	86	231	45	13
0	1	2	3	4	5	6	7	8	9	10
										
2	3	4	5	6	7	8	9	10	11	12

Representação de Strings

- **Strings** são conjuntos de **caracteres**:

```
printf("imprime numero:%d", i);
```

- Representado em código C por texto dentro de aspas duplas “”
 - Cada caractere é codificado como um inteiro de 8 bits (código **ASCII**)
-
- Se imaginarmos que o bit de sinal não é utilizado, cada caractere que compõe uma **string** pode ser representado por um **unsigned char**
 - Inteiro no intervalo [0, 255]
 - Os inteiros de [0, 127] representam os caracteres do **código ASCII padrão**
 - Números e letras “normais”
 - Os inteiros de [128, 255] representam os caracteres do código **ASCII estendido**
 - Letras com acentuação, etc.

Representação de Strings

Tabela ASCII

VALOR DECIMAL	VALOR HEXA-DECIMAL	CONTROL CARACT.	CARACT.	VALOR DECIMAL	VALOR HEXA-DECIMAL	CARACT.	VALOR DECIMAL	VALOR HEXA-DECIMAL	CARACT.	VALOR DECIMAL	VALOR HEXA-DECIMAL	CARACT.	VALOR DECIMAL	VALOR HEXA-DECIMAL	CARACT.	VALOR DECIMAL	VALOR HEXA-DECIMAL	CARACT.
000	00	NUL		043	2B	+	086	56	V	129	81	ü	172	AC	¼	215	D7	#
001	01	SOH	☺	044	2C	,	087	57	W	130	82	é	173	AD	í	216	D8	≠
002	02	STX	☺	045	2D	.	088	58	X	131	83	ô	174	AE	«	217	D9	┘
003	03	ETX	♥	046	2E	:	089	59	Y	132	84	ä	175	AF	»	218	DA	┘
004	04	EOT	♦	047	2F	/	090	5A	Z	133	85	å	176	BO	☐	219	DB	■
005	05	ENQ	♣	048	30	0	091	5B	[134	86	ä	177	B1	☐	220	DC	▬
006	06	ACK	♠	049	31	1	092	5C	\	135	87	ç	178	B2	☐	221	DD	▬
007	07	BEL	•	050	32	2	093	5D]	136	88	è	179	B3		222	DE	▬
008	08	BS	◼	051	33	3	094	5E	^	137	89	ë	180	B4	—	223	DF	▬
009	09	HT	○	052	34	4	095	5F	_	138	8A	è	181	B5	—	224	E0	○
010	0A	LF	☉	053	35	5	096	60	`	139	8B	ï	182	B6	—	225	E1	β
011	0B	VT	♂	054	36	6	097	61	a	140	8C	î	183	B7	—	226	E2	Γ
012	0C	FF	♀	055	37	7	098	62	b	141	8D	ï	184	B8	—	227	E3	Π
013	0D	CR	♪	056	38	8	099	63	c	142	8E	Ä	185	B9	—	228	E4	Σ
014	0E	SO	🎵	057	39	9	100	64	d	143	8F	Å	186	BA		229	E5	σ
015	0F	SI	☼	058	3A	:	101	65	e	144	90	É	187	BB	┘	230	E6	μ
016	10	DLE	▶	059	3B	;	102	66	f	145	91	æ	188	BC	┘	231	E7	τ
017	11	DC1	◀	060	3C	<	103	67	g	146	92	Æ	189	BD	┘	232	E8	φ
018	12	DC2	↑	061	3D	=	104	68	h	147	93	ó	190	BE	┘	233	E9	⊖
019	13	DC3		062	3E	>	105	69	i	148	94	ö	191	BF	┘	234	EA	Ω
020	14	DC4	¶	063	3F	?	106	6A	j	149	95	ò	192	C0	┘	235	EB	δ
021	15	NAK	§	064	40	@	107	6B	k	150	96	ú	193	C1	┘	236	EC	∞
022	16	SYN	—	065	41	A	108	6C	l	151	97	û	194	C2	┘	237	ED	∅
023	17	ETB	↓	066	42	B	109	6D	m	152	98	ÿ	195	C3	┘	238	EE	€
024	18	CAN	↑	067	43	C	110	6E	n	153	99	Ö	196	C4	—	239	EF	∩

024	18	CAN	↑	067	43	C	110	6E	n	153	99	Ö	196	C4	—	239	EF	∩
025	19	EM	↓	068	44	D	111	6F	o	154	9A	Ü	197	C5	+	240	FO	≡
026	1A	SUB	→	069	45	E	112	70	p	155	9B	é	198	C6	≡	241	F1	±
027	1B	ESC	←	070	46	F	113	71	q	156	9C	£	199	C7	≡	242	F2	≥
028	1C	FS	┘	071	47	G	114	72	r	157	9D	¥	200	C8	┘	243	F3	≤
029	1D	GS	↔	072	48	H	115	73	s	158	9E	Þ	201	C9	┘	244	F4	↑
030	1E	RS	▲	073	49	I	116	74	t	159	9F	f	202	CA	┘	245	F5	↓
031	1F	US	▼	074	4A	J	117	75	u	160	A0	ó	203	CB	┘	246	F6	÷
032	20	SP	Space	075	4B	K	118	76	v	161	A1	ì	204	CC	┘	247	F7	≈
033	21		!	076	4C	L	119	77	w	162	A2	ô	205	CD	┘	248	F8	°
034	22		"	077	4D	M	120	78	x	163	A3	ù	206	CE	┘	249	F9	•
035	23		#	078	4E	N	121	79	y	164	A4	ñ	207	CF	┘	250	FA	·
036	24		\$	079	4F	O	122	7A	z	165	A5	Ñ	208	D0	┘	251	FB	√
037	25		%	080	50	P	123	7B	{	166	A6	º	209	D1	┘	252	FC	∩
038	26		&	081	51	Q	124	7C		167	A7	°	210	D2	┘	253	FD	?
039	27		'	082	52	R	125	7D	}	168	A8	¿	211	D3	┘	254	FE	*
040	28		(083	53	S	126	7E	~	169	A9	—	212	D4	┘	255	FF	
041	29)	084	54	T	127	7F	☐	170	AA	—	213	D5	┘			
042	2A		*	085	55	U	128	80	Ç	171	AB	½	214	D6	┘			

Representação de Strings

- Uma **string** é armazenada em **bytes consecutivos** de memória
- Para identificar o final de uma string, a linguagem C utiliza um caractere especial: `'\0'` (código ASCII zero)
- Exemplo:
 - Seja a string “Linguagem C”
 - Imagine a representação desta string na memória como:



String

Posições seguintes não usadas

Cada quadrado representa uma posição de memória de 8 bits.

Representação de Strings

- Imagine a declaração da variável texto como:

```
char texto[100];
```

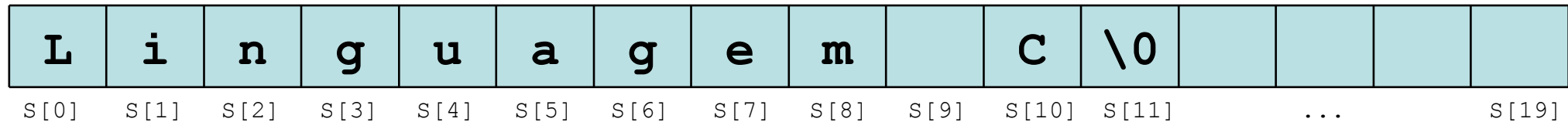
- Se n é uma constante inteira, então os símbolos $[n]$ após o nome da variável indicam que ela poderá ocupar até **n posições de memória consecutivas**
- Logo, a variável texto poderá ocupar até 100 posições do tipo **char**
 - ou seja, 100 bytes
- Como as posições de memória são **consecutivas**, cada uma delas pode ser identificada por um **índice**

Representação de Strings

- Na linguagem C, os valores dos índices começam sempre em **zero**
- Exemplo:
 - Considere que a representação de **S** pode ser imaginada como:

```
char S[20] = "Linguagem C";
```

- Observe que S pode ocupar até 20 posições (numeradas de 0 a 19).



Representação de Strings

- A memória alocada para uma variável é dada por:
(número de posições de memória) * (tamanho do tipo, em bytes)

- Exemplo:

```
int a;  
short int b[15];  
float c[20];  
char d[100] = "dcc-pds1";
```

Variável	Tipo	Nº de posições	Memória alocada
a	int (4 bytes)	1	4 bytes
b	short int (2 bytes)	15	30 bytes
c	float (4 bytes)	20	80 bytes
d	char (1 byte)	100	100 bytes

Representação de Strings

Alocação de memória

- Atenção!

- Uma variável pode ocupar menos memória do que o total de posições alocadas:

```
char S[100] = "Linguagem C";
```

- Dos 100 bytes alocados, a variável está ocupando apenas 12

- Lembre-se do caractere '\0'

- Uma variável jamais poderá ocupar **mais memória** do que o total de memória alocada!

```
double v[50];
```

```
v[49] = 3.33;
```

```
v[50] = 4.33;
```

```
v[51] = 5.33;
```



Memória alocada: 50*8 bytes = 400 bytes

Posições variam de 0 a 49



Representa invasão de memória!

Imprimindo Strings

- Podemos usar:
 - Um laço e imprimir cada um dos elementos como **char**
 - Usando o tag `%c` do `printf`
 - Usar tag `%s` do `printf` diretamente

```
int main() {  
    char str[] = "Minha string";  
    printf("Usando um laço:\t");  
    for (int i = 0; str[i] != '\0'; i++) {  
        printf("%c", str[i]);  
    }  
    printf("\n");  
    printf("usando %%s:\t%s\n", str);  
    return 0;  
}
```



```
./imprimir_str  
Usando um laço: Minha string  
usando %s: Minha string
```

Representação de Strings

Exemplo

```
int main() {  
    char c = 'A';  
    char str[] = "OLA!";  
  
    printf("%s\n", str);  
    return 0;  
}
```

Endereço	Variável	Conteúdo
0xFF1		
0xFF2		
0xFF3		
0xFF4		
0xFF5		
0xFF6		
0xFF7		
0xFF8		
0xFF9		
0xFFA		

Representação de Strings

Exemplo

```
int main() {  
    char c = 'A';  
    char str[] = "OLA!";  
  
    printf("%s\n", str);  
    return 0;  
}
```

Endereço	Variável	Conteúdo
0xFF1	c	'A' ou 65
0xFF2		
0xFF3		'O'
0xFF4		'L'
0xFF5		'A'
0xFF6		'!'
0xFF7		
0xFF8		
0xFF9	str	0xFF3
0xFFA		

Representação de Strings

Exemplo

Sempre que uma string é criada o compilador automaticamente adiciona o carácter especial de “final de string” `\0`

```
int main() {  
    char c = 'A';  
    char str[] = "OLA!";  
  
    printf("%s\n", str);  
    return 0;  
}
```



Como `printf` sabe que chegou no final da **string**?

Endereço	Variável	Conteúdo
0xFF1	c	'A' ou 65
0xFF2		
0xFF3		'O'
0xFF4		'L'
0xFF5		'A'
0xFF6		'!'
0xFF7		
0xFF8		
0xFF9	str	0xFF3
0xFFA		

Representação de Strings

Exemplo

- O que esta acontecendo?

- Ao definir uma `string` usando o comando:

```
char str[] = "OLA!";
```

- Estamos:

- `str` **é um ponteiro!**

- Aponta ao inicio da memoria onde esta o conteúdo da string

- Separando 5 chars em memoria: ``O``, ``L``, ``A`` e ``!``

- Mas esses são só 4!

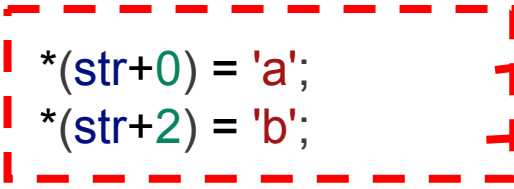
- O compilador automaticamente adiciona `\0` para finalizar a string

- Se não tem `\0` não é uma string!

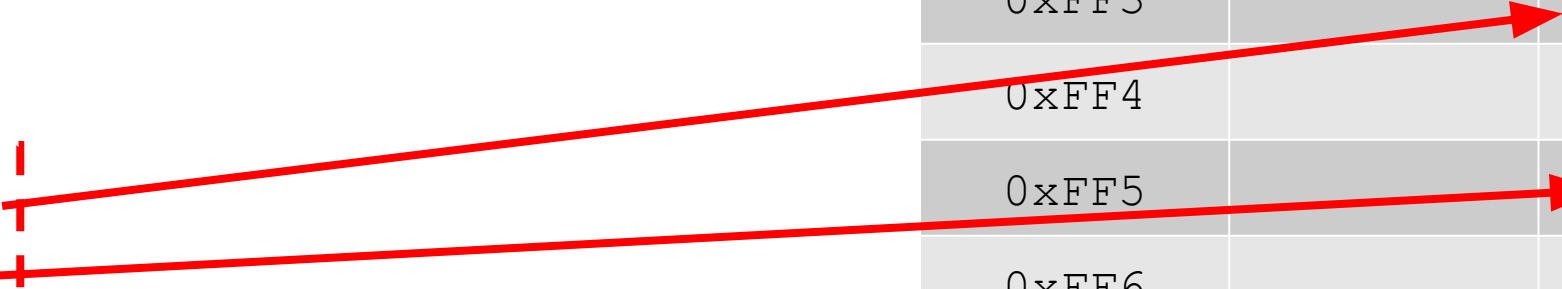
- Só um **arranjo de caracteres**

Representação de Strings

Exemplo

```
int main() {  
    char c = 'A';  
    char str[] = "OLA!";  
      
    *(str+0) = 'a';  
    *(str+2) = 'b';  
    printf("%s\n", str);  
    return 0;  
}
```

Endereço	Variável	Conteúdo
0xFF1	c	'A' ou 65
0xFF2		
0xFF3		'O'
0xFF4		'L'
0xFF5		'A'
0xFF6		'!'
0xFF7		\0
0xFF8		
0xFF9	str	0xFF3
0xFFA		

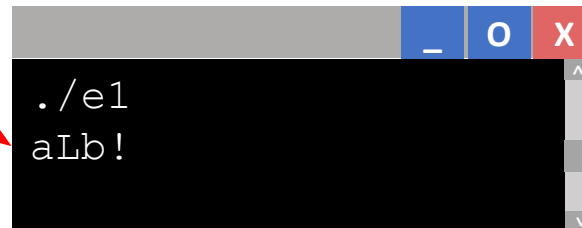


Representação de Strings

Exemplo

- Como sabemos **quantos caracteres andar?**
 - Pela definição do **char** sabemos que ele pula 1 byte (8 bits)

```
int main() {  
    char c = 'A';  
    char str[] = "OLA!";  
  
    *(str+0) = 'a';  
    *(str+2) = 'b';  
  
    printf("%s\n", str);  
    return 0;  
}
```



```
./e1  
aLb!
```

Endereço	Variável	Conteúdo
0xFF1	c	'A' ou 65
0xFF2		
0xFF3		'a'
0xFF4		'L'
0xFF5		'b'
0xFF6		'!'
0xFF7		\0
0xFF8		
0xFF9	str	0xFF3
0xFFA		

Representação de Strings

Exemplo

- O que esta acontecendo?

- Ao acessar uma `string` usando o comando:

`*(str+2) = 'a';`

- Estamos:

- `str` é um ponteiro ao inicio do texto (exemplo `0xFF5500`)
- Acessando o endereço de memoria do `str`, e pulando “duas casas”
 - `0xFF5500 + 2 = 0xFF5502`
- A posição `0xFF5502` é atualizada com uma nova letra, usando o operador `*` para acessar o **valor** e assignando a letra `'a'`
- Fazer contas dessa forma é complicado, tem uma forma melhor?
 - **Sim!** Acesso via índices `[]`

Representação de Strings

Exemplo

```
int main() {  
    char c = 'A';  
    char str[] = "OLA!";  
  
    *(str + 0) = 'a';  
    *(str + 2) = 'b';  
    printf("%s\n", str);  
  
    str[1] = 'Z';  
    str[2] = '\0';  
    printf("%s\n", str);  
  
    return 0;  
}
```

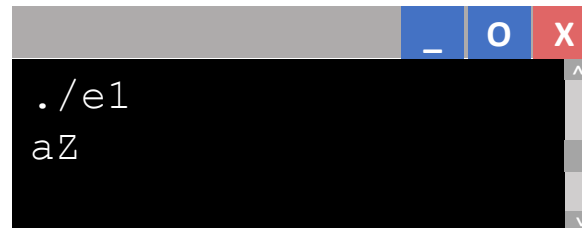
Endereço	Variável	Conteúdo
0xFF1	c	'A' ou 65
0xFF2		
0xFF3		'a'
0xFF4		'L'
0xFF5		'b'
0xFF6		'!'
0xFF7		\0
0xFF8		
0xFF9	str	0xFF3
0xFFA		

Representação de Strings

Exemplo

```
int main() {  
    char c = 'A';  
    char str[] = "OLA!";  
  
    *(str + 0) = 'a';  
    *(str + 2) = 'b';  
    printf("%s\n", str);  
  
    str[1] = 'Z';  
    str[2] = '\0';  
    printf("%s\n", str);  
  
    return 0;  
}
```

Endereço	Variável	Conteúdo
0xFF1	c	'A' ou 65
0xFF2		
0xFF3		'a'
0xFF4		'z'
0xFF5		\0
0xFF6		'!'
0xFF7		\0
0xFF8		
0xFF9	str	0xFF3
0xFFA		



```
./e1  
aZ
```

Representação de Strings

Exemplo

- O que esta acontecendo?

- Ao executar:

```
str[2] = '\0';  
printf("%s\n", str);
```

- Estamos:

- Acessando o endereço de memória do `str`, e pulando “duas casas”
 - Usando o acesso via índice `[x]`
 - Assignando o valor `'\0'` que é o “fim de uma string”
 - Ao imprimir com `printf` ele para no primeiro `'\0'` que achar:
 - Vai imprimir `"aZ"`
 - Ao invés de `"aZ!"`

Endereço	Variável	Conteúdo
0xFF3		'a'
0xFF4		'z'
0xFF5		\0
0xFF6		'!'
0xFF7		\0
0xFF9	str	0xFF3

Endereços de String

```
int main() {  
    char str[] = "Linguagem C";  
    printf("%s\n", str);  
  
    printf("%c-%c-%c-%c\n",  
        str[0],  
        *str,  
        *str + 1,  
        *(str + 1));  
  
    for (int i = 0; str[i] != '\0'; i++) {  
        printf("endereço de str[%d] (%c): %p ou %p\n",  
            i,  
            str[i],  
            &str[i],  
            str + i);  
    }  
  
    return 0;  
}
```

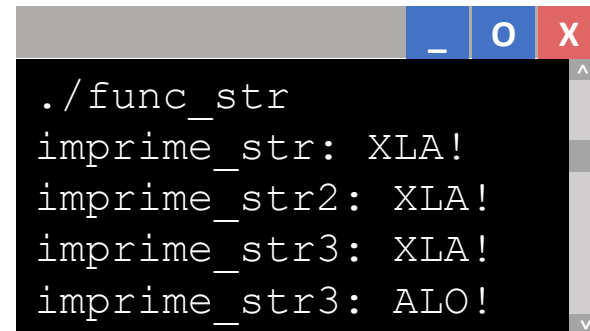
```
./enderecos  
Linguagem C  
L-L-M-i  
  
end. de str[0] (L): 0x16b3c6b88 ou 0x16b3c6b88  
end. de str[1] (i): 0x16b3c6b89 ou 0x16b3c6b89  
end. de str[2] (n): 0x16b3c6b8a ou 0x16b3c6b8a  
end. de str[3] (g): 0x16b3c6b8b ou 0x16b3c6b8b  
end. de str[4] (u): 0x16b3c6b8c ou 0x16b3c6b8c  
end. de str[5] (a): 0x16b3c6b8d ou 0x16b3c6b8d  
end. de str[6] (g): 0x16b3c6b8e ou 0x16b3c6b8e  
end. de str[7] (e): 0x16b3c6b8f ou 0x16b3c6b8f  
end. de str[8] (m): 0x16b3c6b90 ou 0x16b3c6b90  
end. de str[9] ( ): 0x16b3c6b91 ou 0x16b3c6b91  
end. de str[10] (C): 0x16b3c6b92 ou 0x16b3c6b92
```


Usando strings em funções

Exemplo

```
int main() {  
    char str[] = "OLA!";  
  
    imprime_str(str);  
    imprime_str2(str);  
    imprime_str3(str);  
  
    // Não compila  
    // (implicit function declarations)  
    // imprime_str1("ALO!");  
  
    // Compila mas da segmentation fault  
    // imprime_str2("ALO!");  
  
    imprime_str3("ALO!");  
    return 0;  
}
```

```
void imprime_str(char str[]) {  
    str[0] = 'X';  
    printf("imprime_str: %s\n", str);  
}  
  
void imprime_str2(char *str) {  
    str[0] = 'X';  
    printf("imprime_str2: %s\n", str);  
}  
  
void imprime_str3(const char *str) {  
    // Não compila, pois str é constante  
    // str[0] = 'X';  
    printf("imprime_str3: %s\n", str);  
}
```



```
./func_str  
imprime_str: XLA!  
imprime_str2: XLA!  
imprime_str3: XLA!  
imprime_str3: ALO!
```

Usando strings em funções

Exemplo

- O que esta acontecendo?
 - `imprime_str(char str[])` e `imprime_str(char *str)`
 - São análogos, recebem um ponteiro (que pode ser modificado)
 - `imprime_str(const char *str)`
 - Recebe um ponteiro “constante”
 - O conteúdo dessa string **não pode ser modificado!**
 - Da erro de compilação!

Recebendo strings como input

- Como podemos solicitar uma string do usuário?
 - Podemos usar o `scanf`

```
int main() {  
    int i = 0;  
    char str[5];  
  
    printf("Insira string: ");  
    for (i = 0; i < 5; i++) {  
        scanf("%c", &str[i]);  
        if (str[i] == '\n') {  
            break;  
        }  
    }  
    printf("\n%s\n", str);  
    return 0;  
}
```

stdin	Conteúdo
0	
1	
2	
3	
4	
5	

Endereço	Variável	Conteúdo
0xFF3		
0xFF4		
0xFF5		
0xFF6		
0xFF7		
0xFF9	str	0xFF3

Recebendo strings como input

- Como podemos solicitar uma string do usuário?
 - Podemos usar o `scanf`

```
int main() {  
    int i = 0;  
    char str[5];  
  
    printf("Insira string: ");  
    for (i = 0; i < 5; i++) {  
        scanf("%c", &str[i]);  
        if (str[i] == '.') {  
            break;  
        }  
    }  
    printf("\n%s\n", str);  
    return 0;  
}
```

stdin	Conteúdo
0	
1	'i'
2	'm'
3	'.'
4	
5	

Endereço	Variável	Conteúdo
0xFF3		
0xFF4		
0xFF5		
0xFF6		
0xFF7		
0xFF9	str	0xFF3

Recebendo strings como input

- Como podemos solicitar uma string do usuário?
 - Podemos usar o `scanf`

```
int main() {  
    int i = 0;  
    char str[5];  
  
    printf("Insira string: ");  
    for (i = 0; i < 5; i++) {  
        scanf("%c", &str[i]);  
        if (str[i] == '.') {  
            break;  
        }  
    }  
    printf("\n%s\n", str);  
    return 0;  
}
```

stdin	Conteúdo
0	
1	'm'
2	'.'
3	
4	
5	

Endereço	Variável	Conteúdo
0xFF3		'S'
0xFF4		
0xFF5		
0xFF6		
0xFF7		
0xFF9	str	0xFF3

Recebendo strings como input

- Como podemos solicitar uma string do usuário?
 - Podemos usar o `scanf`

```
int main() {  
    int i = 0;  
    char str[5];  
  
    printf("Insira string: ");  
    for (i = 0; i < 5; i++) {  
        scanf("%c", &str[i]);  
        if (str[i] == '.') {  
            break;  
        }  
    }  
    printf("\n%s\n", str);  
    return 0;  
}
```

stdin	Conteúdo
0	
1	\.'
2	
3	
4	
5	

Endereço	Variável	Conteúdo
0xFF3		'S'
0xFF4		'i'
0xFF5		
0xFF6		
0xFF7		
0xFF9	str	0xFF3

Recebendo strings como input

- Como podemos solicitar uma string do usuário?
 - Podemos usar o `scanf`

```
int main() {  
    int i = 0;  
    char str[5];  
  
    printf("Insira string: ");  
    for (i = 0; i < 5; i++) {  
        scanf("%c", &str[i]);  
        if (str[i] == '.') {  
            break;  
        }  
    }  
    printf("\n%s\n", str);  
    return 0;  
}
```

stdin	Conteúdo
0	.
1	
2	
3	
4	
5	

Endereço	Variável	Conteúdo
0xFF3		'S'
0xFF4		'i'
0xFF5		'm'
0xFF6		
0xFF7		
0xFF9	str	0xFF3

Recebendo strings como input

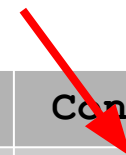
- Como podemos solicitar uma string do usuário?
 - Podemos usar o `scanf`

```
int main() {  
    int i = 0;  
    char str[5];  
  
    printf("Insira string: ");  
    for (i = 0; i < 5; i++) {  
        scanf("%c", &str[i]);  
        if (str[i] == '.') {  
            break;  
        }  
    }  
    printf("\n%s\n", str);  
    return 0;  
}
```

stdin	Conteúdo
0	
1	
2	
3	
4	
5	

Endereço	Variável	Conteúdo
0xFF3		'S'
0xFF4		'i'
0xFF5		'm'
0xFF6		'.'
0xFF7		
0xFF9	str	0xFF3

Isso aqui é uma string?
Esta faltando o `'\0'`



Recebendo strings como input

- Como podemos solicitar uma string do usuário?
 - Podemos usar o `scanf`

```
int main() {  
    int i = 0;  
    char str[5];  
  
    printf("Insira string: ");  
    for (i = 0; i < 5; i++) {  
        scanf("%c", &str[i]);  
        if (str[i] == '.') {  
            break;  
        }  
    }  
    str[i] = '\0';  
    printf("%s\n", str);  
    return 0;  
}
```

Isso aqui é uma string?
Esta faltando o `'\0'`

stdin	Conteúdo
0	
1	
2	
3	
4	
5	

Endereço	Variável	Conteúdo
0xFF3		'S'
0xFF4		'i'
0xFF5		'm'
0xFF6		' '
0xFF7		'\0'
0xFF9	str	0xFF3

Recebendo strings como input

Outras formas

- `scanf("%s", str);`
 - Vai pegar a string até o primeiro espaço
 - Se digitar "oi tudo bem?" a string vai conter só "oi"
 - **Cuidado!** "tudo bem?" ainda esta no buffer do `stdin`
 - Isso pode gerar bugs no seu programa!
- `fgets(str, sizeof(str), stdin);`
 - Vai pegar a string até o primeiro salto de linha ou preencher o tamanho da string
- **Importante:**
 - A função `gets()` (sem o `f`) captura uma string, **mas não deve ser usada**
 - Ela é **deprecada** pois permite que usuários abusem do tamanho da string, inserindo cadeias de texto de qualquer tamanho!

Problema I

- Uma empresa quer transmitir mensagens sigilosas a seus diretores e precisa de um programa para codificar suas mensagens
- A regra de criptografia deve substituir o código ASCII de cada caractere que compõe a mensagem por:
 - $(5 * \text{código_ASCII} + 100) \% 256$
- As mensagens deverão terminar com '.' (ponto)

Problema I

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

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

```
int main() {  
    int i, n;  
    char optei;  
    char texto[100];  
  
    do {  
        printf("Entre com o texto a ser codificado:\n");  
        for (i = 0; i < 100; i++) {  
            scanf("%c", &texto[i]);  
            if (texto[i] == '.')  
                break;  
        }  
  
        n = i;  
        printf("Texto codificado:\n");  
        for (i = 0; i < n; i++) {  
            char tmp = texto[i];  
            texto[i] = (5 * texto[i] + 100) % 256;  
            printf("%c\t(%d)\t->\t%c\t(%d)\n", tmp, tmp, texto[i], texto[i]);  
        }  
        printf("\n\nContinua S/N: ");  
    } while (optei == 'S');  
    return 0;  
}
```

```
./p1  
Entre com o texto a ser codificado:  
bebe 123.  
Texto codificado:  
b      (98)      ->      N      (78)  
e      (101)     ->      ]      (93)  
b      (98)      ->      N      (78)  
e      (101)     ->      ]      (93)  
                (32)      ->      (4)  
1      (49)      ->      Y      (89)  
2      (50)      ->      ^      (94)  
3      (51)      ->      c      (99)  
  
Continua S/N:
```

Entradas de texto usando strings

- Tamanho de um string: **strlen**
- Copiar um string: **strcpy, strncpy**
 - Funções com **n** no nome recebem o tamanho do arranjo como parâmetro
 - Evita erros caso o string não caiba no arranjo
- Concatenar strings: **strcat, strncat**
 - String de destino tem que ter espaço
- Comparar strings: **strcmp, strncmp**
 - Comparar strings com **==** só compara a posição dos arranjos, não o conteúdo
- Duplicar um string: **strdup**

Utilizando *strings*

- Podemos fazer laços para operar sobre os caracteres de um *string*
 - Abordagem tediosa 😊
- Biblioteca padrão do C tem dezenas de funções de manipulação de strings

```
#include <string.h>
```

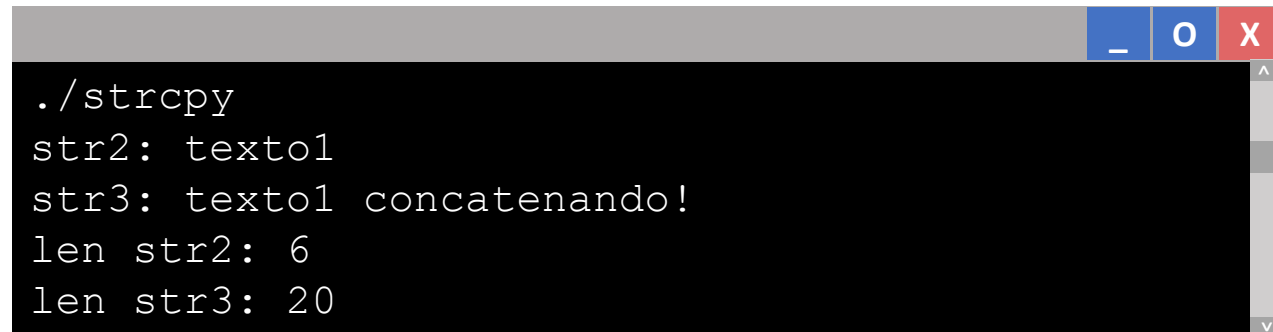
Manipulação de strings

- Tamanho de um string: **strlen**
- Copiar um string: **strcpy, strncpy**
 - Funções com **n** no nome recebem o tamanho do arranjo como parâmetro
 - Evita erros caso o string não caiba no arranjo
- Concatenar strings: **strcat, strncat**
 - String de destino tem que ter espaço
- Comparar strings: **strcmp, strncmp**
 - Comparar strings com **==** só compara a posição dos arranjos, não o conteúdo
- Duplicar um string: **strdup**

Manipulação de strings

Copia, concatenação e tamanho

```
int main() {  
    char str1[] = "texto1";  
    char str2[40];  
    strcpy(str2, str1);  
  
    char str3[40];  
    strcpy(str3, str2);  
    strcat(str3, " concatenando!");  
  
    printf("str2: %s\nstr3: %s\n", str2, str3);  
    printf("len str2: %lu\nlen str3: %lu\n",  
           strlen(str1),  
           strlen(str3));  
    return 0;  
}
```



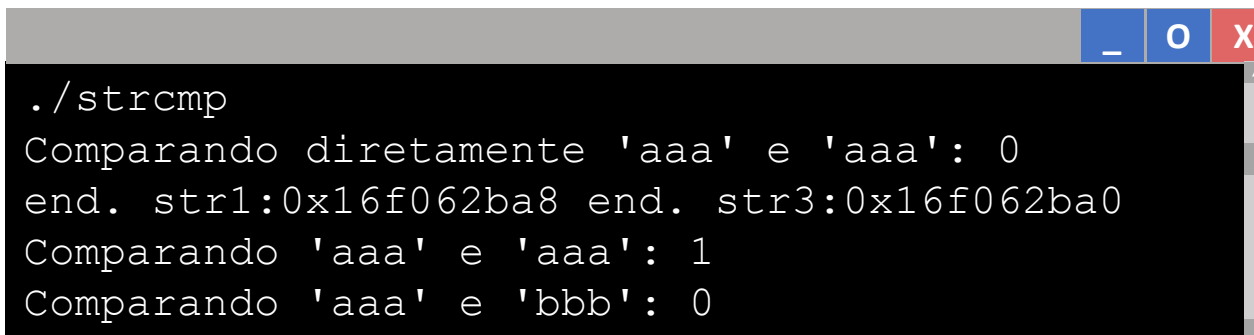
A terminal window with a black background and white text. The window has a title bar with a minus sign, a maximize button, and a close button. The output of the program is as follows:

```
./strcpy  
str2: texto1  
str3: texto1 concatenando!  
len str2: 6  
len str3: 20
```


Manipulação de strings

Comparação

```
int main() {  
    char str1[] = "aaa";  
    char str2[] = "bbb";  
    char str3[] = "aaa";  
  
    printf("Comparando diretamente '%s' e '%s': %d\n", str1, str3, str1 == str3);  
    printf("end. str1:%p end. str3:%p\n", str1, str3);  
    printf("Comparando '%s' e '%s': %d\n", str1, str3, strcmp(str1, str3) == 0);  
    printf("Comparando '%s' e '%s': %d\n", str1, str2, strcmp(str1, str2) == 0);  
    return 0;  
}
```



A terminal window with a dark background and white text. The window has a title bar with a minus sign, a maximize button, and a close button. The output of the program is as follows:

```
./strcmp  
Comparando diretamente 'aaa' e 'aaa': 0  
end. str1:0x16f062ba8 end. str3:0x16f062ba0  
Comparando 'aaa' e 'aaa': 1  
Comparando 'aaa' e 'bbb': 0
```

Conversão de strings

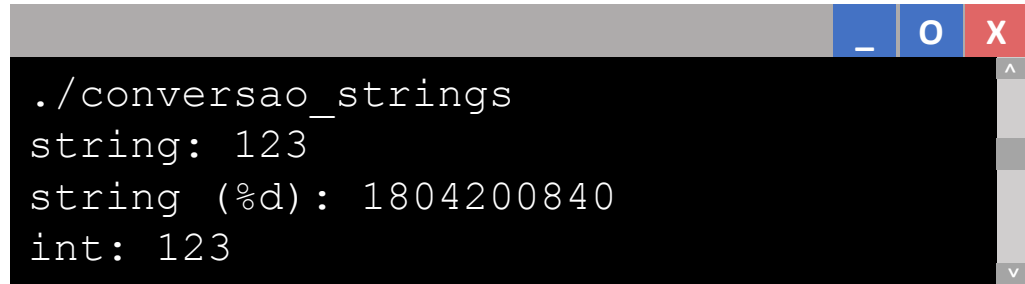
- Converter string para inteiros:

- `int atoi(const char *string)`
- `long atol(const char *string)`
- `long long atoll(const char *string)`

- Converter de string para ponto flutuante

- `double atof(const char *string)`

```
int main() {  
    char str[] = "123";  
    int novo_int = atoi(str);  
  
    printf("string: %s\n", str);  
    printf("string (%d): %d\n", str);  
    printf("int: %d\n", novo_int);  
  
    return 0;  
}
```



```
./conversao_strings  
string: 123  
string (%d): 1804200840  
int: 123
```

Considerações finais

- **Strings** são muito uteis! Mas com muito poder tem muitas responsabilidades:

1. Cuidado com o **tamanho** da string:
 - A memória reservada para sua string tem que ser maior ou igual aos dados que você vai inserir nela
2. Verificar criteriosamente o **final da string**
 - Onde o primeiro caráter “\0” estiver, é o final da string
3. **Usar as bibliotecas** de string sempre que possível
 - Não reinventar a roda!

Perguntas?

- E-mail:
 - hector@dcc.ufmg.br
- Material da disciplina:
 - <https://pedroolmo.github.io/teaching/pdsI.html>
- Github:
 - <https://github.com/h3ct0r>



Héctor Azpúrua
h3ct0r