Estruturas de Dados

Profa. Juliana Pinheiro Campos

Vetores

- Forma mais simples de estruturar um conjunto de dados.
  - Exemplo:

```
int p[10];          // vetor de inteiros com 10 elementos
```

Reserva de um espaço de memória de 40 bytes, pois cada int ocupa 4 bytes.

Vetores

- O acesso a cada elemento é feito por meio de uma indexação.
  - A indexação de um vetor em C varia de 0 a n-1, onde n é a dimensão do vetor;

p[0] → Acessa o primeiro elemento do vetor p;
p[1] → Acessa o segundo elemento do vetor p;
...
p[9] → Acessa o último elemento do vetor p;

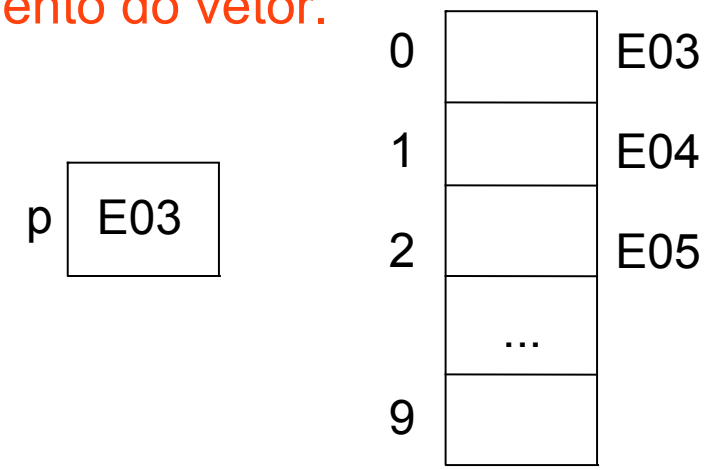
v[10] → ERRO!!! (invasão de memória)

Vetores

- Existe uma forte associação entre vetores e ponteiros.

  int p[10] → o símbolo p (que representa o vetor) é uma constante que representa seu endereço inicial.

p sem índice aponta para o primeiro elemento do vetor.



ESTRUTURAS DE DADOS

- Ponteiros
  - C suporta aritmética de ponteiros desde que o valor do ponteiro resultante aponte para dentro da área reservada para o vetor.
  - Operações válidas: adição e subtração.

```
int a = 5;
int *p = &a;
p++;
```

Como p é um ponteiro para inteiro, p++ ou p + 1 representa um ponteiro para o próximo inteiro na memória (valor de p incrementado em 4 bytes)

Passagem de vetores para funções

- Passar um vetor para uma função consiste em passar o endereço da primeira posição do vetor.
- A função chamada deve ter um parâmetro do tipo ponteiro para armazenar esse valor.
- Exemplo: 8) cálculo da média de 10 valores.

  OBS: Como é passado o endereço do primeiro elemento do vetor, os valores dos elementos podem ser alterados dentro da função.

ESTRUTURAS DE DADOS

Vetores

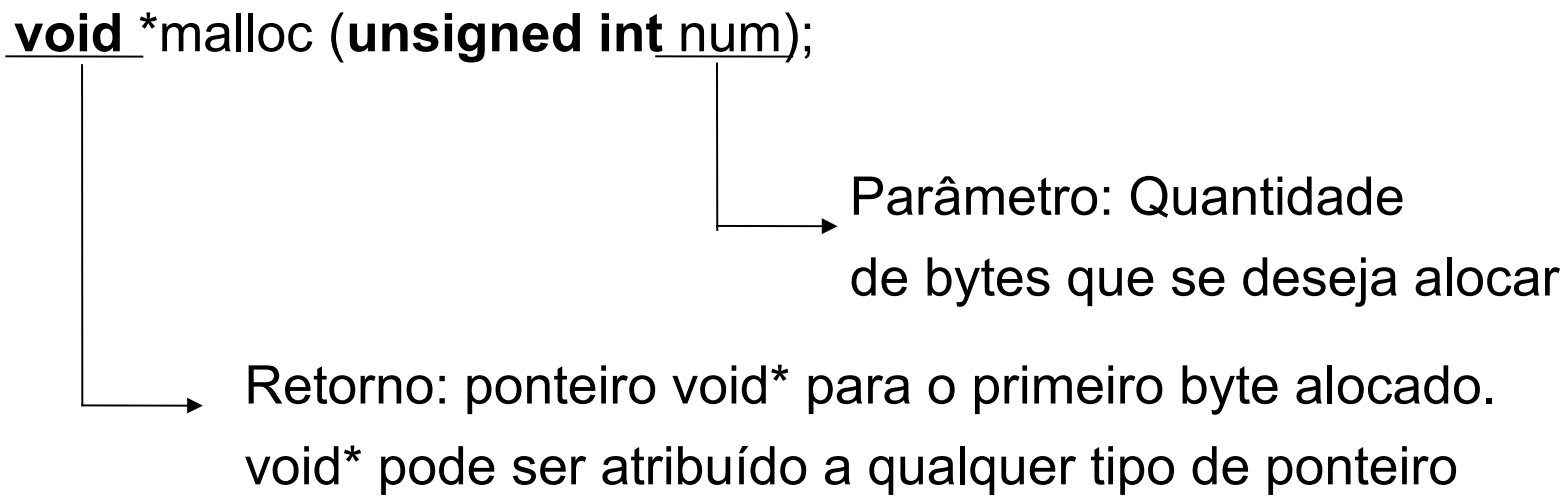
- Limitação de vetor: ter que dimensioná-lo na declaração, ou seja, durante a codificação.
- Solução: Alocação dinâmica.

Alocação dinâmica de memória

- Requisitar ao sistema,em tempo de execução, um espaço de um determinado tamanho.
- Esse espaço permanece reservado até que seja liberado explicitamente pelo programa.
- Se o espaço de memória livre for menor do que o espaço requisitado dinamicamente, a alocação não é feita.

Ponteiros e alocação dinâmica de memória

- Função para alocação dinâmica de memória
  - malloc (presente na biblioteca stdlib)



Ponteiros e alocação dinâmica de memória

- Exemplo de alocação dinâmica de memória

void *malloc (**unsigned int** num);

Alocação de um vetor de inteiros com 10 elementos:

```
int* p;
p = malloc(10*4);
```

Para ficar independente de compiladores e máquinas, usamos o operador sizeof.

```
p = malloc(10*sizeof(int));
```

Ponteiros e alocação dinâmica de memória

- Exemplo de alocação dinâmica de memória

void *malloc (**unsigned int** num);

Alocação de um vetor de inteiros com 10 elementos fazendo conversão explícita:

```
p = (int*) malloc(10 * sizeof(int));
```

Ponteiros e alocação dinâmica de memória

- Se não houver espaço livre suficiente para realizar a alocação, a função retorna um endereço nulo (NULL definido em stdlib.h)

- É comum verificar o valor de retorno da função malloc para cercar esse tipo de erro.

Ponteiros e alocação dinâmica de memória

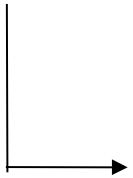
- Exemplo de alocação dinâmica de memória

```
float *p = NULL;
p = (float *) malloc(1 * sizeof(float));
*p = -3.2
```

Ponteiros e alocação dinâmica de memória

- Função para liberar memória alocada dinamicamente
  - free (presente na biblioteca stdlib)

void free (**void *p**);



Parâmetro: ponteiro da memória a ser liberado.

Ponteiros e alocação dinâmica de memória

- Exemplo de liberação de memória

```
int *p = NULL;
p = (float *) malloc(1 * sizeof(float));
*p = -3.2;
free(p);
```

OBS: Todo ponteiro que aponta para área alocada dinamicamente, deve liberar o espaço após o seu uso.

Exemplo: 9) cálculo da média, usando alocação dinâmica.

Ponteiros e alocação dinâmica de memória

- Funções podem ter um ponteiro como tipo de retorno.
  - Exemplo

```
int main(){
        int *p;
        p = inicializaValores();
}
int* inicializaValores() {
    int *v = (int*) malloc(5 * sizeof(int));
    int i;
    for(i=0; i<5; i++) {
            v[i] = i+10;
        printf("\nv[%i]: %i", i, v[i]);
    }
    return v;
}
```

Atenção: Se uma função retorna um vetor, a alocação do vetor deve ser dinâmica!

Ponteiros e alocação dinâmica de memória

- Função para realocar um vetor dinamicamente:
  - realloc (presente na biblioteca stdlib)

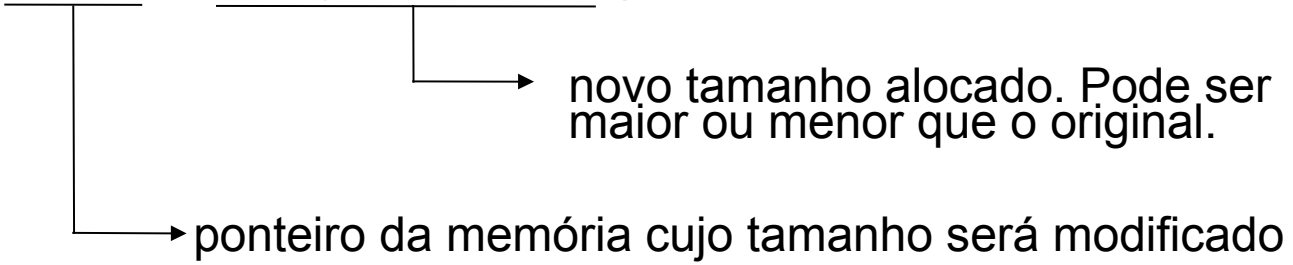
    - Redimensiona um espaço alocado previamente em tempo de execução.

    - Faz realocação do vetor preservando o conteúdo dos elementos que permanecem válidos após a realocação.

Ponteiros e alocação dinâmica de memória

- Função para realocar um vetor dinamicamente:
  - realloc (presente na biblioteca stdlib)

void* **realloc** (void *ptr, unsigned int num);



Exemplo:

```
v = realloc(v, num * sizeof(int));
```

Ponteiros e alocação dinâmica de memória

- Vetores bidimensionais: **Matrizes estáticas**
  - C permite a criação de vetores bidimensionais, declarados estaticamente.
  - Declaração de uma matriz:

tipo nome[num de linhas] [num de colunas];

  - Exemplo:

```
float mat[4][3];
```

Reserva espaço de memória para os 12 elementos da matriz de forma contínua

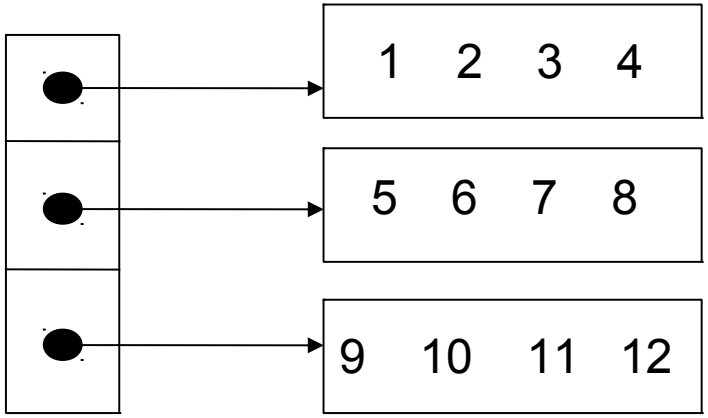
ESTRUTURAS DE DADOS

- Vetores bidimensionais: **Matrizes dinâmicas**
  - C também permite a criação de vetores declarados dinamicamente: Matriz representada por um vetor de ponteiros
  - Cada linha da matriz é representada por um vetor independente.
  - A matriz é representada por um vetor de vetores (vetor de ponteiros) onde cada elemento armazena o endereço do primeiro elemento de cada linha.

ESTRUTURAS DE DADOS

- Vetores bidimensionais: **Matrizes dinâmicas**
  - Representada por um vetor de ponteiros

    Ex: Matriz 3 x 4



Ponteiros e alocação dinâmica de memória

- Vetores bidimensionais: **Matrizes dinâmicas**
  - Declaração de variáveis: Matriz(3x2)

```
int i, j; //tipo: int
int *v1=NULL, *v2=NULL, *v3=NULL; // tipo: ponteiro para int
int **matriz=NULL; // ponteiro para ponteiro para int
```

  - Alocação de memória

```
v1 = (int *) malloc(2 * sizeof(int));
v2 = (int *) malloc(2 * sizeof(int));
v3 = (int *) malloc(2 * sizeof(int));
matriz = (int **) malloc(3 * sizeof(int *));
```

Ponteiros e alocação dinâmica de memória

- Matriz(3x2)
  - Atribuição de valores à matriz;

```
matriz[0] = v1;
matriz[1] = v2;
matriz[2] = v3;
```

  - Eliminando referências

```
free(v1);
free(v2);
free(v3);
free (matriz);
```

Libera espaço de memória alocada dinamicamente e elimina as referências

Ponteiros e alocação dinâmica de memória

- Matriz(3x2)
  - Atribuição de valores

Forma: matriz[i][j] = valor;

Exemplo: matriz[2][1] = -1;

Ponteiros e alocação dinâmica de memória

- Vetores bidimensionais: **Matrizes dinâmicas**
  - Alocação de memória: Matriz (m x n)

```
int i;
int **matriz;


matriz = (int **)malloc(m * sizeof(int*));
for(i = 0; i < m; i++){
        matriz[i] = (int*) malloc(n * sizeof(int));
}
```

Ponteiros e alocação dinâmica de memória

- Vetores bidimensionais: **Matrizes dinâmicas**
  - Liberação de memória: Matriz (m x n)

```
for(i = 0; i < m; i++)
  free(matriz[i]);
free(matriz);
```

Exercício

- Implemente uma função que, dada uma matriz, crie dinamicamente a matriz transposta correspondente cujo protótipo é:

```
int** transposta(int m, int n, int** mat);
```

Uma matriz Q é a matriz transposta de M, se

$Q_{ij} = M_{ji}$, para qualquer elemento da matriz.

Cadeia de caracteres (String)

- String é a palavra utilizada para referenciar uma cadeia (sequência) de caracteres;
- C não oferece o tipo caracter.
- Os caracteres são representados na memória do computador por códigos numéricos.
- Char = 1 byte = 8 bits.
- Com 8 bits tem-se 256 combinações de 0 e 1.

  Exemplos: 00000000, 00000001, …
- É possível representar valores de -128 a 127;

Cadeia de caracteres (String)

- A correspondência entre os caracteres e seus códigos numéricos é feita por uma tabela de código (Ex: ASCII – American Standard Code Information Interchange)

- Em geral, usa-se a tabela ASCII, mas diferentes máquinas podem usar diferentes códigos.

Cadeia de caracteres (String)

Códigos associados a alguns caracteres da tabela ASCII

	0	1	2	3	4	5	6	7	8	9
30			sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	S	t	u	v	w
120	x	y	z	{	|	}	~			

Cadeia de caracteres (String)

- A diferença entre caracteres e inteiros depende da forma como eles são tratados.

- Exemplo

```
char c = 97;
printf("%d %c\n",c,c);
```

Imprime o mesmo valor usando modificador de formato diferente. Imprime: 97 a

Cadeia de caracteres (String)

- A linguagem C permite a escrita de constantes caracteres usando aspas simples.
- A expressão 'a' representa uma constante caracter e resulta no valor numérico associado ao caractere a.
- O uso de constantes caracteres nos livra de ter de conhecer os códigos associados a cada caractere.

```
char c = 'a';
printf("%d %c\n",c,c);
```

Cadeia de caracteres (String)

- Exemplo: Função que testa se caractere c é um dígito.

```
int digito(char c) {

    if ((c>='0')&&(c<='9'))
        return 1;
    else
        return 0;

}
```

→ Na tabela ASCII, os dígitos, as letras maiúsculas e minúsculas são codificados em sequência.

Como seria uma função que verifica se determinado caractere representa uma letra?
E uma função que converte uma letra minúscula em maiúscula?

Cadeia de caracteres (String)

- String: São representadas por vetores do tipo char.
- Devem ser terminados pelo caractere nulo: ‘\0’

- Exemplo

```
#include <stdio.h>
int main() {
        char nome[10];
        scanf("%s", nome);    //Juliana
        printf("%s\n", nome);
}
```

‘J’
‘u’
‘l’
i
‘a’
‘n’
‘a’
‘\0’
‘z’
‘x’

Cadeia de caracteres (String)

- Todas as funções que manipulam cadeias de caracteres recebem como parâmetro um vetor de char e processam caractere por caractere até encontrar o caractere nulo.
- Vantagem de ter o caractere nulo: não é necessário passar o número de caracteres para as funções.
- Exemplo: o especificador de formato %s da função printf permite imprimir uma cadeia de caracteres.

Cadeia de caracteres (String)

- Exemplo de inicialização

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

As duas inicializações de cidade estão corretas

Leitura de caracteres

- Utilizamos a função scanf com o especificador de formato %c para ler um caractere fornecido via teclado.
- O especificador de formato %c não pula caracteres brancos (' ', '\t', '\n'). Para fazer isso, inclua um espaço em branco antes do especificador de formato.
- A função getchar também pode ser utilizada para ler um caractere.

```
Ex:     char c;
c = getchar();
```

Leitura de strings

- Utilizamos a função scanf com o especificador de formato %s para ler uma cadeia de caracteres.
- O especificador de formato %s pula os caracteres brancos e captura uma sequência de caracteres não brancos.
- Isso significa que o especificador de formato %s captura apenas nomes simples.
- Para ler nomes compostos, podemos usar o especificador de formato %[...] no qual listamos entre os colchetes todos os caracteres que aceitaremos na leitura.

Leitura de strings

- Ex: %[aeiou] – lê uma sequência de vogais, até encontrar um caractere que não seja uma vogal.
- Se o primeiro caractere entre colchetes for ^, teremos o efeito inverso.
- Ex: %[^\n] – lê uma sequência de caracteres até encontar '\n'.
- Um número após o símbolo % limita o número máximo de caracteres.

Ex: char cidade[81];

scanf(“ %80[^\n]”, cidade);

Cadeia de caracteres (String)

- Funções que manipulam Strings
  - Função que imprime uma cadeia de caracteres:

```
void imprime (char* s) {
        int i;
        for (i=0; s[i] != '\0'; i++)
                printf("%c",s[i]);
        printf("\n");
}
```

Cadeia de caracteres (String)

- Função que retorna o número de caracteres existentes na cadeia:

```
int comprimento (char* s) {

        int i;

        int n = 0;

        for (i=0; s[i] != '\0'; i++)
                n++;

        return n;
}
```

Cadeia de caracteres (String)

- Função para copiar os elementos de uma cadeia de caracteres para outra.

```
void copia (char* dest, char* orig) {
        int i;
        for (i=0; orig[i] != '\0'; i++)
                dest[i] = orig[i];
        dest[i] = '\0';
}
```

A função supõe que dest tem espaço suficiente para realizar a operação.

Cadeia de caracteres (String)

- Função para concatenar uma cadeia de caracteres com outra, ou seja, os caracteres de uma cadeia são copiados no final da outra cadeia.

```
void concatena (char*dest, char* orig) {
                    int i = 0;
                    int j;

                    i = 0;
                    while (dest[i] != '\0')
                            i++;

                    for (j=0; orig[j] != '\0'; j++) {
                            dest[i] = orig[j];
                            i++;
                    }

                    dest[i] = '\0';
}
```

Cadeia de caracteres (String)

- Função para comparar 2 cadeias de caracteres. Usa códigos numéricos associados para determinar a ordem relativa entre eles. Retorna -1 se a 1ª cadeia precede a 2ª, 1 se a 2ª preceder a 1ª e 0 se as 2 forem iguais.

```
int compara(char* s1, char* s2) {
                int i;

                for (i=0; s1[i] != '\0' && s2[i] != '\0'; i++) {
                        if(s1[i] < s2[i])
                                return -1;
                        else if(s1[i] > s2[i])
                                return 1;
        }
                if(s1[i] == s2[i])
                        return 0;

                else if(s2[i] != '\0')
                        return -1;
                else
                        return 1;
}
```

Cadeia de caracteres (String)

- Funções que manipulam Strings (definidas na biblioteca string.h)
  - strlen (comprimento): recebe como parâmetro de entrada uma cadeia de caracteres e fornece como retorno o número de caracteres existentes na cadeia.
  - strcpy (cópia): copia os elementos de uma cadeia de caracteres para outra.
  - strcat (concatena): concatena uma cadeia de caracteres com outra já existente.
  - strcmp (compara): compara duas cadeias dadas. Retorna -1 se a 1ª cadeia precede a 2ª, 1 se a 2ª preceder a 1ª e 0 se as cadeias tiverem a mesma sequencia de caracteres.

Cadeia de caracteres (String)

- Funções que manipulam Strings (definidas na biblioteca string.h)
  - Exemplo com alocação dinâmica:

```
#include <string.h>

char* duplica (char* s) {
                int n = strlen(s);
                char* d = (char*) malloc ((n+1)*sizeof(char));
                strcpy(d,s);
                return d;
}
```

ESTRUTURAS DE DADOS

Funções recursivas que manipulam Strings

- Uma cadeia de caracteres pode ser definida de forma recursiva. Uma cadeia de caracteres é representada por:

  - Uma cadeia de caracteres vazia;
  - Um caractere seguido de uma sub(cadeia) de caracteres.

- Podemos dizer que uma cadeia s não-vazia pode ser representada pelo seu primeiro caractere s[0] seguido da cadeia que começa no endereço do segundo caractere, &s[1].

ESTRUTURAS DE DADOS

- Função recursiva para imprimir uma cadeia de caracteres:

```
void imprime_rec (char* s) {

        if (s[0] != '\0') {
                printf("%c",s[0]);
                imprime_rec(&s[1]);
        }
}
```

Como alterar a função para imprimir os caracteres da cadeia na ordem inversa?

ESTRUTURAS DE DADOS

- Função recursiva que retorna o número de caracteres existentes em uma cadeia:

```
int comprimento_rec (char* s) {

        if (s[0] == '\0')
                return 0;
        else
                return 1 + comprimento_rec(&s[1]);
        }
```