Algoritmos e Estruturas de Dados I
Aula 9.1 - Ponteiro e Arranjos

Prof. Felipe Lara

Ponteiros x Arranjos

- Ponteiros também podem ser usados para acessar arranjos.
- Considere o seguinte arranjo de inteiros:

```
int myNumbers[4] = {25, 50, 75, 100};

for (int i = 0; i < 4; i++) {
  printf("%d\n", myNumbers[i]);
}
```

```
//Imprime:
25
50
75
100
```

Ponteiros x Arranjos

- Ponteiros também podem ser usados para acessar arranjos.
- Considere o seguinte arranjo de inteiros:

```
int myNumbers[4] = {25, 50, 75, 100};

for (int i = 0; i < 4; i++) {
  printf("%d\n", myNumbers[i]);
}
```

```
//Imprime:
25
50
75
100
```

```
int myNumbers[4] = {25, 50, 75, 100};

for (int i = 0; i < 4; i++) {
  printf("%p\n", &myNumbers[i]);
}
```

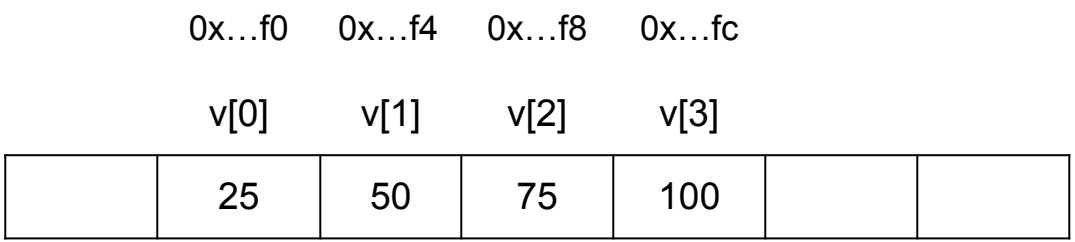
```
//Imprime:
0x7ffe70f9d8f0
0x7ffe70f9d8f4
0x7ffe70f9d8f8
0x7ffe70f9d8fc
```

Ponteiros x Arranjos

```
int v[4] = {25, 50, 75, 100};

for (int i = 0; i < 4; i++) {
  printf("%p\n", &v[i]);
}
```

```
//Imprime:
0x7ffe70f9d8f0
0x7ffe70f9d8f4
0x7ffe70f9d8f8
0x7ffe70f9d8fc
```

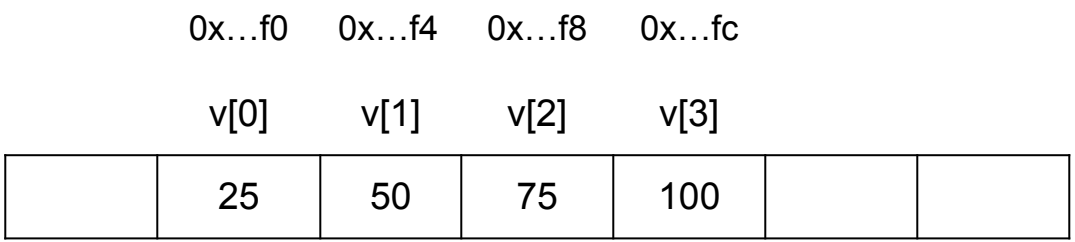


Ponteiros x Arranjos

```
int v[4] = {25, 50, 75, 100};

for (int i = 0; i < 4; i++) {
  printf("%p\n", &v[i]);
}
```

```
//Imprime:
0x7ffe70f9d8f0
0x7ffe70f9d8f4
0x7ffe70f9d8f8
0x7ffe70f9d8fc
```



Logo, podemos imprimir (percorrer) o arranjo com ponteiro!

Ponteiros x Arranjos

- Em C, a variável que define arranjo é, na verdade, um ponteiro para o primeiro elemento do arranjo.
- O endereço do primeiro elemento do arranjo é o mesmo da variável do arranjo.

Ponteiros x Arranjos

- Em C, a variável que define arranjo é, na verdade, um ponteiro para o primeiro elemento do arranjo.
- O endereço do primeiro elemento do arranjo é o mesmo da variável do arranjo.
- Considere o seguinte arranjo de inteiros:

```
int myNumbers[4] = {25, 50, 75, 100};

printf("%p\n", myNumbers);

printf("%p\n", &myNumbers[0])}
```

```
//Imprime:
0x7ffe70f9d8f0
0x7ffe70f9d8f0
```

Ponteiros x Arranjos

- Imprimindo o vetor com ponteiro:

```
int myNumbers[4] = {25, 50, 75, 100};
int *ptr = myNumbers;

for (int i = 0; i < 4; i++) {
  printf("%d\n", *(ptr + i));
}
```

```
//Imprime:
25
50
75
100
```

```
int myNumbers[4] = {25, 50, 75, 100};
int *ptr = myNumbers;

for (int i = 0; i < 4; i++) {
  printf("%d\n", ptr[i]);
}
```

```
//Imprime:
25
50
75
100
```

Ponteiros x Arranjos

- Preenchendo o vetor com ponteiro:

```
int myNumbers[4];
int *ptr = myNumbers;

for (int i = 0; i < 4; i++) {
  scanf("%d", ptr+i);
}
```

```
int myNumbers[4];
int *ptr = myNumbers;

for (int i = 0; i < 4; i++) {
  scanf("%d", &ptr[i]);
}
```

Ponteiros x Arranjos

- Preenchendo o vetor com ponteiro:

```
int myNumbers[4];
int *ptr = myNumbers;

*ptr = 10;
*(ptr+1) = 20;
*(ptr+2) = 30;
*(ptr+3) = 40;
```

```
int myNumbers[4];
int *ptr = myNumbers;

ptr[0] = 10;
ptr[1] = 20;
ptr[2] = 30;
ptr[3] = 40;
```

Ponteiros

- Exercício: O que será impresso pelo programa?

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

int main()
{
    const int TAM = 10;
    int vet[10];
    int *ptr = vet;
    for(int i = 0; i < TAM; i++) {
        *(ptr) = (i+1)*(i+1);
        ptr++;
    }
    for(int i = 0; i < TAM; i++) {
        ptr--;
        printf("%d ", *(ptr));
    }
}
```

Ponteiros

- Exercício: O que será impresso pelo programa?

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

int main()
{
    const int TAM = 10;
    int vet[10];
    int *ptr = vet;
    for(int i = 0; i < TAM; i++) {
        *(ptr) = (i+1)*(i+1);
        ptr++;
    }
    for(int i = 0; i < TAM; i++) {
        ptr--;
        printf("%d ", *(ptr));
    }
}
```

Resposta:

100 81 64 49 36 25 16 9 4 1

Ponteiros

- Exercício: Considere a matriz a seguir. Como imprimir a matriz usando ponteiro?

```
#include <stdio.h>


int main()
{
    int mat[2][3];
}
```

Ponteiros

- Exercício: Considere a matriz a seguir. Como imprimir a matriz usando ponteiro?

```
#include <stdio.h>


int main()
{
    int mat[2][3];
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 3; j++)
            scanf("%d", &mat[i][j]);
    int *p = &mat[0][0];
    for (int i = 0; i < 6; i++)
        printf("%d ", *(p + i));
    return 0;
}
```

Alocação Dinâmica

Alocação Dinâmica de Memória

- A área de alocação dinâmica é chamada heap. A área de alocação estática é chamada stack.

	Stack (Alocação estática - automática)	**Heap (Alocação dinâmica - manual)**
Controle	Feita automaticamente pelo compilador	Feita manualmente pelo programador (ex: malloc, new)
Tempo de vida	Dura enquanto a função está sendo executada	Dura até ser liberada explicitamente (ou pelo coletor de lixo, em linguagens como Java/Python)
Acesso	Mais rápido (endereços contíguos e previsíveis)	Mais lento (endereços aleatórios na memória)
Tamanho	Limitado (pode causar *stack overflow*)	Muito maior, limitado apenas pela RAM disponível
Uso típico	Variáveis locais, parâmetros de função	Estruturas grandes, objetos criados em tempo de execução

Alocação Dinâmica de Memória

- A área de alocação dinâmica é chamada heap. A área de alocação estática é chamada stack.

- A linguagem C oferece um conjunto de funções que permitem a alocação ou a liberação dinâmica de memória:
  - malloc(): aloca memória e não inicializa
  - calloc(): aloca memória e inicializa com bits zerados
  - realloc(): altera o tamanho da memória alocada
  - free(): libera a memória alocada

- As funções estão disponíveis na biblioteca “stdlib.h”.

- As funções trabalham com ponteiros.

Alocação Dinâmica de Memória

- **malloc:**
  - a função “malloc” ou “alocação em memória” em C é usada para alocar dinamicamente um único bloco de memória com o tamanho especificado.

```
int *ptr;
int n = 5;
ptr = (int*) malloc(n * sizeof(int));
```

- Serão alocados 20 bytes por cada inteiro ter tamanho de 4 bytes.
- Se não houver espaço disponível, malloc retorna NULL.

Alocação Dinâmica de Memória

- **calloc:**
  - a função “calloc” aloca memória dinamicamente, mas inicializa todos os bits com zero.

```
int *ptr;
int n = 5;
ptr = (int*) calloc(n, sizeof(int));
```

- Serão alocados 20 bytes por cada inteiro ter tamanho de 4 bytes.
- Se não houver espaço disponível, calloc retorna NULL.

Alocação Dinâmica de Memória

- **realloc:**
  - a função de “realocação” em C realloc é usada para alterar dinamicamente uma alocação feita anteriormente com “malloc”.
  - A realocação de memória mantém a informação já armazenada e os blocos extras alocados serão inicializados com “lixo”. Se não houver espaço, retorna NULL.

```
int *pi;
pi = (int *) malloc(sizeof(int));
pi = (int *) realloc(pi, 5*sizeof(int));
```

Alocação Dinâmica de Memória

- **free:**
  - A função free em C é usada para desalocar dinamicamente a memória.
  - A memória alocada com as funções malloc (), calloc() e realloc () não é desalocada automaticamente.
  - A função free é ajuda a reduzir o desperdício de memória ao liberá-la

```
int *pi;
pi = (int *) malloc(sizeof(int));
pi = (int *) realloc(pi, 5*sizeof(int));
free(pi);
```

Alocação Dinâmica de Memória

Exemplo: Programa para calcular a soma de n números digitados pelo usuário.

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

int main() {
  int n, i, *ptr, sum = 0;

  printf(“Digite a quantidade de números: ");
  scanf("%d", &n);

  //alocando memória
  ptr = (int*) malloc(n * sizeof(int));

  //se a memória não tiver sido alocada
  if(ptr == NULL) {
    printf("Erro na alocação dinâmica");
    exit(0);
  }
  printf(“Digite os números: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }

  printf("Soma = %d", sum);

  //desalocando memória
  free(ptr);

  return 0;
}
```

Alocação Dinâmica de Memória

Exemplo: Programa para calcular a soma de n números digitados pelo usuário.

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

int main() {
  int n, i, *ptr, sum = 0;

  printf("Digite a quantidade de números: ");
  scanf("%d", &n);

  //alocando memória
  ptr = (int*) malloc(n * sizeof(int));

  //se a memória não tiver sido alocada
  if(ptr == NULL) {
    printf("Erro na alocação dinâmica");
    exit(0);
  }

  printf("Digite os números: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }

  printf("Soma = %d", sum);

  //desalocando memória
  free(ptr);

  return 0;
}
```

```
Digite a quantidade de números: 3
Digite os números: 100
20
36
Soma = 156
```

Alocação Dinâmica de Memória

Exercício: Crie um programa que aloca dinamicamente uma matriz com NLIN linhas e NCOL colunas, onde NLIN e NCOL são definidos pelo usuário. O programa deve preencher a matriz e imprimir em seguida.

```c
//Solução 1 - Usando ponteiro simples

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int NLIN, NCOL, *p;
    printf("Digite o numero de linhas e colunas: ");
    scanf("%d %d", &NLIN, &NCOL);
    p = (int*) malloc(NLIN * NCOL * sizeof(int));
    for(int i = 0; i < NLIN * NCOL; i++) {
        printf("Digite um numero: ");
        scanf("%d", p+i); //scanf("%d", &p[i]);
    }
    for(int i = 0; i < NLIN; i++) {
        for(int j = 0; j < NCOL; j++) {
            printf("%5d", *(p + i*NCOL + j)); //printf("%5d", p[i*NCOL + j]);
        }
        printf("\n");
    }
    free(p);
    return 0;
}
```

```c
// Solução 2 - Usando ponteiro simples
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int NLIN, NCOL, *p;
    printf("Digite o numero de linhas e colunas: ");
    scanf("%d %d", &NLIN, &NCOL);
    p = (int*) malloc(NLIN * NCOL * sizeof(int));
    for(int i = 0; i < NLIN * NCOL; i++) {
        printf("Digite um numero: ");
        scanf("%d", p+i); //scanf("%d", &p[i]);
    }
    for(int i = 0; i < NLIN * NCOL; i++) {
        if(i % NCOL == 0)
            printf("\n");
        printf("%5d", *(p + i)); //printf("%5d", p[i]);
    }
    free(p);
    return 0;
}
```

//Solução 3 – Usando arranjo de ponteiro

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

int main()
{
    int NLIN, NCOL;
    printf("Digite o numero de linhas e colunas: ");
    scanf("%d %d", &NLIN, &NCOL);
    int *p[NLIN];
    for(int i = 0; i < NLIN; i++)
        p[i] = (int*) malloc(NCOL * sizeof(int));
    for(int i = 0; i < NLIN; i++) {
        for(int j = 0; j < NCOL; j++) {
            printf("Digite um numero: ");
            scanf("%d", &p[i][j]); //scanf("%d", p[i] + j);
        }
    }
    for(int i = 0; i < NLIN; i++) {
        for(int j = 0; j < NCOL; j++)
            printf("%5d", p[i][j]); //printf("%5d", *(p[i] + j));
        printf("\n");
    }
    for(int i = 0; i < NLIN; i++)
        free(p[i]);
    return 0;
}
```

Solução 4 – Usando ponteiro para ponteiro

```
int main()
{
    int NLIN, NCOL;
    scanf("%d %d", &NLIN, &NCOL);
    int **p = (int**) malloc(NLIN * sizeof(int*));
    for(int i = 0; i < NLIN; i++)
        p[i] = (int*) malloc(NCOL * sizeof(int));
    for(int i = 0; i < NLIN; i++) {
        for(int j = 0; j < NCOL; j++)
            scanf("%d", &p[i][j]); //scanf("%d", *(p+i) + j);
    }
    for(int i = 0; i < NLIN; i++) {
        for(int j = 0; j < NCOL; j++)
            printf("%5d", p[i][j]); //printf("%5d", *(*(p+i) + j));
        printf("\n");
    }
    for(int i = 0; i < NLIN; i++)
        free(p[i])
    free(p);
    return 0;
}
```

PUC Minas

Dúvidas?