Introdução a Programação

Ponteiros e Vetores, Alocação Dinâmica

Tópicos da Aula

- Hoje aprenderemos que existe uma forte relação entre ponteiros e vetores
  - Associação entre ponteiros e vetores
  - Ponteiros constantes x Ponteiros variáveis
  - Passagem de ponteiros invés de vetores para funções
  - Comando sizeof
  - Alocação dinâmica
  - Vetores dinâmicos

Associação entre Vetores e Ponteiros

- Considere a declaração:

```
int   v [10] ;
```

  - O símbolo v
    - Representa o vetor
    - É uma constante que representa seu endereço inicial
    - Aponta para o primeiro elemento do vetor

Ponteiros e Vetores (matrizes)

- Em C existe um relacionamento muito forte entre ponteiros e vetores
  - O compilador entende todo vetor e matriz como ponteiros, pois a maioria dos computadores é capaz de manipular ponteiros e não vetores
  - Qualquer operação que possa ser feita com índices de um vetor pode ser feita com ponteiros
  - O identificador de um vetor representa um endereço, ou seja, um ponteiro

Ponteiros e Vetores

- Como vimos, C permite aritmética de ponteiros
- Se tivermos a declaração

```
int   v [10] ;
```

- Podemos acessar elementos do vetor através de aritmética de ponteiros

`v + 0` ⟶ Aponta para (igual ao endereço do) primeiro elemento do vetor

`v + 1` ⟶ Aponta para o segundo elemento do vetor

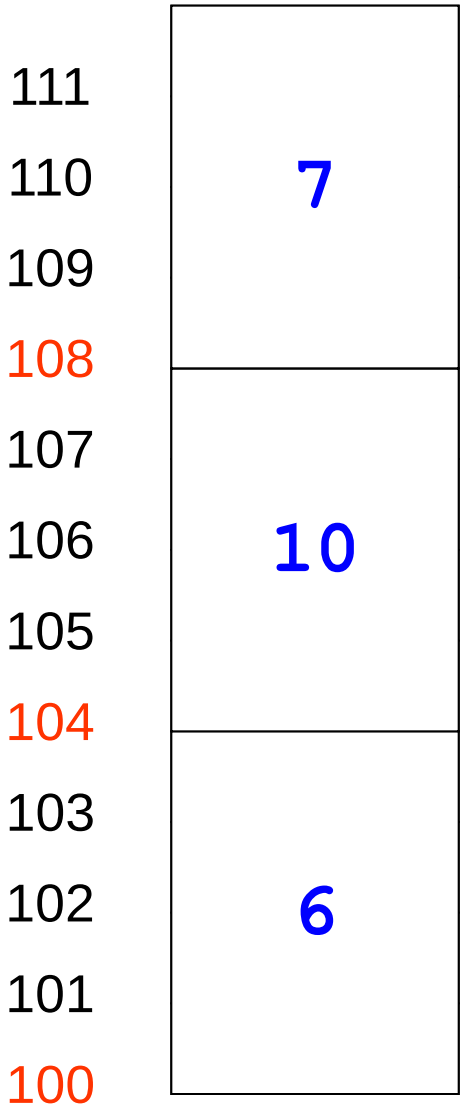
⋮

`v + 9` ⟶ Aponta para o último elemento do vetor

- Portanto: `&v[i]↔(v + i)` `v[i]↔ *(v + i)`

Representando Ponteiros e Vetores na Memória

Memória




```
int v[] = {6,10,7};
```

```
*(v + 2) ↔ v[2] ↔ 7
```

```
v + 2 ↔ &v[2] ↔ 108
```

```
*(v + 1) ↔ v[1] ↔ 10
```

```
v + 1 ↔ &v[1] ↔ 104
```

```
*v ↔ v[0] ↔ 6
```

```
v  ↔ &v[0] ↔ 100
```

Ponteiros e Vetores

Vetores podem ser tratados como ponteiros em C!

```
int a[10];
int *pa;
pa = &a[0];
pa = a;
```

→

```
*pa ↔ a[0] ↔ pa[0]
*(pa+i) ↔ a[i] ↔
pa[i] ↔ *(a+i)
a+i ↔ &a[i]
```

Expressões Equivalentes!

Usando Notação de Ponteiros para Vetores

Versão com Vetor

```
int main( ) {
    int nums[ ] = {1, 4, 8};
    int cont;
    for(cont=0; cont < 3; cont++) {
        printf("%d\n,nums[cont]);
    }
 }
```

Versão com Ponteiro

```
int main( ) {
    int nums[ ] = {1, 4, 8};
    int cont;
    for(cont=0; cont < 3; cont++) {
        printf("%d\n,*(nums + cont));
    }
 }
```

Ponteiros Constantes x Ponteiros Variáveis

Declaração de uma constante do tipo ponteiro para inteiros (ponteiro constante)

```
int main( ) {
    int nums[ ] = {1, 4, 8};
    int cont;
    for(cont=0; cont < 3; cont++) {
       printf("%d\n",*(nums++));
    }
 }
```

Errado!

Tenta incrementar endereço constante `nums` e atualizar a constante com novo endereço

Ponteiros Constantes x Ponteiros Variáveis

```
int main( ) {
    int nums[ ] = {1, 4, 8};
    int* pnums = nums;
    int cont;
    for(cont=0; cont < 3; cont++) {
       printf("%d\n,*(pnums++));
    }
 }
```

Declaração de uma variável do tipo ponteiro para inteiros (ponteiro variável)

Certo!

Incrementa endereço armazenado na variável pnums e atualiza a variável com novo endereço

Ponteiros Constantes x Ponteiros Variáveis

```
int a[10];
int *pa;
pa = a;
```

Atribui a uma variável um novo endereço: CERTO!

```
int a[10];
int *pa;
a = pa;
```

Atribui a uma constante um novo endereço: ERRADO!

Passando Vetores como Argumentos para Funções

```
#include <stdio.h>
float  media(int n, float num[]){
  int i;
  float s = 0.0;
  for(i = 0; i < n; i++)
       s  = s +  num[i] ;
  return  s/n ;
}
int main(){
     float numeros[10] ;
     float med;
     int i ;
     for(i = 0; i < 10; i++)
          scanf ("%f", &numeros[i]) ;
     med  =  media(10, numeros ) ;
     ...
}
```

Parâmetro do tipo vetor de `float`

Endereço inicial do vetor é passado como argumento

Passando Ponteiros invés de Vetores como Argumentos para Funções

```
#include <stdio.h>
float  media(int n, float* num){
  int i;
  float s = 0.0;
  for(i = 0; i < n; i++)
       s  = s +  num[i] ;
  return  s/n ;
}
int main(){
     float numeros[10] ;
     float med;
     int i ;
     for(i = 0; i < 10; i++)
          scanf ("%f", &numeros[i]) ;
     med  =  media(10, numeros ) ;
     ...
}
```

Parâmetro do tipo ponteiro para float

Endereço inicial (ponteiro) do vetor é passado como argumento

Passando Ponteiros como Argumentos de Funções

- Considere a seguinte assinatura de função:

```
void incrementa(int n, int* v)
```

Pergunta: Parâmetro `v` é um ponteiro para um vetor de inteiros ou para uma variável do tipo inteiro?

Resposta 1: Não tem como saber

Resposta 2: É indiferente. Podemos considerar um ponteiro para uma variável do tipo inteiro como um ponteiro para um vetor com um só elemento

Comando *sizeof*

- Forma Geral:

```
sizeof(tipo) ou sizeof(variavel)
```

- Informa o número de bytes de um dado tipo ou variável em **tempo de compilação**
- Exemplo:

`int d = sizeof(float);` → d armazena o valor 4

Usando sizeof para Determinar Tamanho de Ponteiros e Vetores

Qual é o o numero de elementos?

3

```
int main() {
   int num[ ]={1,2,3};
   int numElementos = sizeof(num)/sizeof(int);
   printf ("Tamanho = %d\n", sizeof(num));
   printf ("Num elementos = %d\n", numElementos);
}
```

Qual é o o numero de elementos?

1

```
int main() {
   int num[ ]={1,2,3};
   int* num2 = num;
   int numElementos = sizeof(num2)/sizeof(int);
   printf ("Tamanho = %d\n", sizeof(num2));
   printf ("Num elementos = %d\n", numElementos);
}
```

Alocação de Memória

- Quando declaramos uma variável, o compilador reserva (aloca) um espaço na memória suficiente para armazenar valores do tipo da variável
  - Alocação estática (em tempo de compilação)

```
int var ;
```

Aloca espaço para 1 `int`

```
char s1 [10];
```

Aloca espaço para 10 `char`

```
char*  s2;
```

Aloca espaço para 1 endereço

Alocação Dinâmica

- Modos de alocar espaço em memória:
  - Estaticamente
    - Variáveis globais (e estáticas): O espaço reservado para a variável existe enquanto o programa estiver sendo executado
    - Variáveis locais: O espaço existe enquanto a função, que declarou a variável, estiver sendo executada.
  - Dinamicamente
    - Requisitar memória em **tempo de execução**: O espaço alocado dinamicamente permanece reservado até que seja ***explicitamente*** liberado pelo programa

Alocação Dinâmica em C

- Função básica para alocar memória é **malloc** presente na biblioteca **stdlib.h**

```
void* malloc(unsigned qtdBytes);
```

  - Recebe como argumento um número inteiro sem sinal que representa a quantidade de bytes que se deseja alocar
  - Retorna o endereço inicial da área de memória alocada.

Alocação Dinâmica em C com `malloc`

- Aloca somente a quantidade de memória necessária
  - Exemplo:

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

Se a alocação for bem sucedida, v armazenará o endereço inicial de uma área contínua de memória suficiente para armazenar 10 valores inteiros (40 bytes)

Alocação Dinâmica em C com `malloc`

- Uso do comando **`sizeof`** para ter independência de plataforma de desenvolvimento
  - Exemplo:

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

- Função *malloc* retorna um ponteiro genérico, para qualquer tipo, representado por *void
  - Faz-se a conversão para o tipo apropriado usando o operador de molde de tipo *(cast)*

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

Erro na Alocação

- Se não houver espaço livre suficiente para realizar a alocação, a função **`malloc`** retorna um **endereço nulo**
  - É representado pelo símbolo **`NULL`**
  - É uma boa prática de programação testar se a alocação foi bem sucedida para evitar erros de execução

Liberando Espaço Alocado

- Uso da função **`free`** para liberar espaço de memória alocada dinamicamente

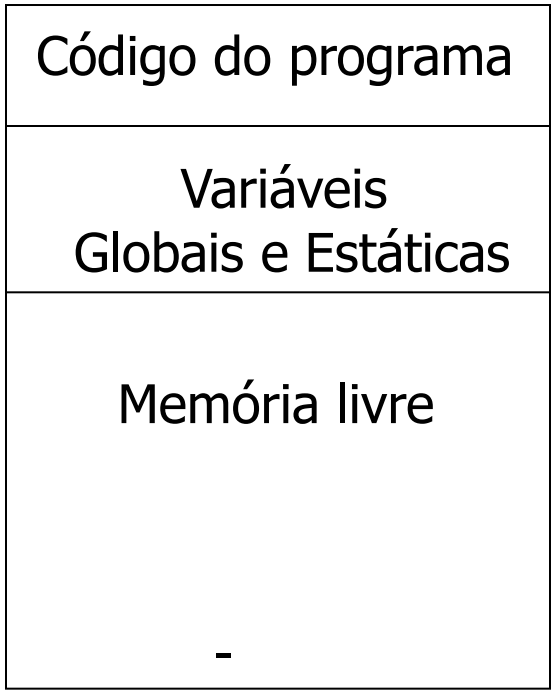
```
void free(void* endereco);
```

  - Recebe como parâmetro o ponteiro da memória a ser liberada
  - O espaço de memória fica livre para ser alocado futuramente pelo próprio programa ou outro programa
  - Recomenda-se liberar espaço de memória previamente alocado que não é mais necessário
    - Evita desperdício

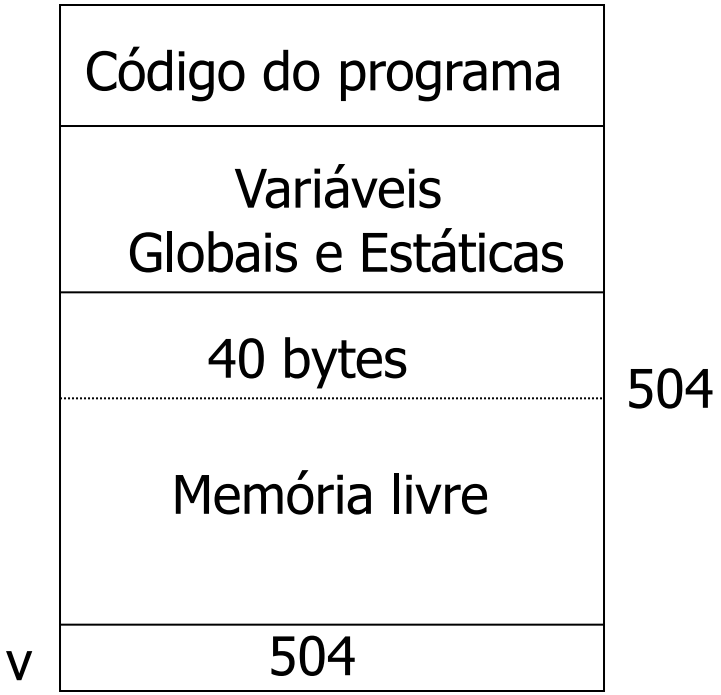
Memória e Alocação Dinâmica

1) Declaração: int *v ; 2) v=(int*)malloc(10*sizeof (int));

Abre-se espaço na pilha para o ponteiro (variável local)



Reserva-se o espaço de memória da área livre e atribui o endereço à v



Usando Alocação Estática para Calcular Média com Vetores

```
#include <stdio.h>
int main() {
    int qtdNumeros,contador = 0;
    float numeros[2000];
    float media = 0.0;
    do{
        printf("Quantidade de numeros? (< 2000):\n");
        scanf("%d", &qtdNumeros);
    } while (qtdNumeros <= 0 || qtdNumeros > 2000);
    while (contador < qtdNumeros) {
        scanf("%f",&numeros[contador]);
        media = media + numeros[contador];
        contador++;
    }
    media = media/qtdNumeros;
    printf("\nA media eh %f\n");
    return 0;
}
```

Declaração estática do tamanho do vetor limita aplicação

Tamanho pode ser insuficiente ou grande demais (desperdício)

Vetores Dinâmicos

- Declaração de vetores implicam em alocação estática de memória
- Com alocação dinâmica, podemos criar algo como um vetor cujo tamanho é decidido em tempo de execução, ou seja um **vetor dinâmico**
- Para tal, usaremos variáveis do tipo ponteiro que receberão os endereços iniciais do espaço alocado dinamicamente
  - Com o endereço inicial, podemos navegar pelo vetor

Usando Alocação Dinâmica para Calcular Média com Vetores Dinâmicos

```
int main() {
    int qtdNumeros,contador = 0;
    float* numeros;
    float media = 0.0;
    do{
        printf("Quantidade de numeros?:\n");
        scanf("%d", &qtdNumeros);
    } while (qtdNumeros <= 0);
    numeros = (float*) malloc(qtdNumeros*sizeof(float));
    if (numeros != NULL) {
        while (contador < qtdNumeros) {
                scanf("%f",&numeros[contador]);
                media = media + numeros[contador];
                contador++;
        }
    } /* continua */
}
```

Ponteiro recebe endereço de espaço alocado dinamicamente (vetor dinâmico)

Tamanho do vetor é determinado pelo usuário

Função Realloc

- Podemos mudar o espaço de memória alocado previamente de forma dinâmica
- Para isso, podemos utilizar a função **realloc**

```
void* realloc(void* ptr,unsigned qtdBytes);
```

  - Recebe endereço do bloco de memória alocado previamente
  - Recebe como argumento um número inteiro sem sinal que representa a quantidade de bytes que se deseja alocar
  - Retorna o endereço inicial da área de memória alocada
    - Se endereço retornado for diferente do passado como parâmetro, conteúdo do bloco original é copiado para novo endereço

Usando Realloc

```
int main() {
    int qtdNumeros = 5,contador = 0;
    char resposta;
    float media = 0.0;
    float* nums, *numsR;
    nums = (float*) malloc(qtdNumeros*sizeof(float));
    if (nums == NULL) {
        printf("Memoria insuficiente");exit(1);
    }
    printf("Programa calcula media de 5 numeros.");
    printf("Deseja mais/menos? (s/n)\n");
    scanf("%c",&resposta);
    if (resposta == 's') {
        printf("Quantidade de numeros?:\n");
        scanf("%d", &qtdNumeros);
        numsR = (float*) realloc(nums,qtdNumeros*sizeof(float));
        if (numsR != NULL) {
                nums = numsR;
        }/* continua */
    }
```

Vetor é alocado dinamicamente

Tamanho do vetor muda dinamicamente

Vetores e Alocação Dinâmica

- Vetores Locais e Funções

```
float*   prod_vetorial (float*  u , float*  v) {
      float  p[3] ;
      p[0] = u [ 1 ] * v [ 2 ]  -  v [ 1 ]  * u [ 2 ] ;
      p[1] = u [ 2 ] * v [ 0 ]  -  v [ 2 ]  * u [ 0 ] ;
      p[2] = u [ 0 ] * v [ 1 ]  -  v [ 0 ]  * u [ 1 ] ;
      return  p  ;
}
```

ERRADO! – Endereço local é retornado

A variável retornada é declarada localmente. Por isso, sua área de memória deixa de ser válida quando a função termina.

Vetores e Alocação Dinâmica

- Vetores Locais e Funções
  - Forma Correta:

```
float*   prod_vetorial (float*  u , float*  v) {
      float*  p = (float*) malloc(3 * sizeof(float)) ;
      p[0] = u [ 1 ] * v [ 2 ]  -  v [ 1 ]  * u [ 2 ] ;
      p[1] = u [ 2 ] * v [ 0 ]  -  v [ 2 ]  * u [ 0 ] ;
      p[2] = u [ 0 ] * v [ 1 ]  -  v [ 0 ]  * u [ 1 ] ;
      return  p  ;
}
```

CERTO! – Endereço alocado dinamicamente fica disponível até que seja liberado explicitamente

Resumindo ...

- Relação entre ponteiros e vetores
- Ponteiros constantes x Ponteiros variáveis
- Passagem de ponteiros invés de vetores para funções
- Comando sizeof
- Alocação dinâmica e Vetores dinâmicos