Alocação Dinâmica de Memória

Algoritmos e Estruturas de Dados I

Prof. Cristiano Rodrigues

Prof. Lucas Astore

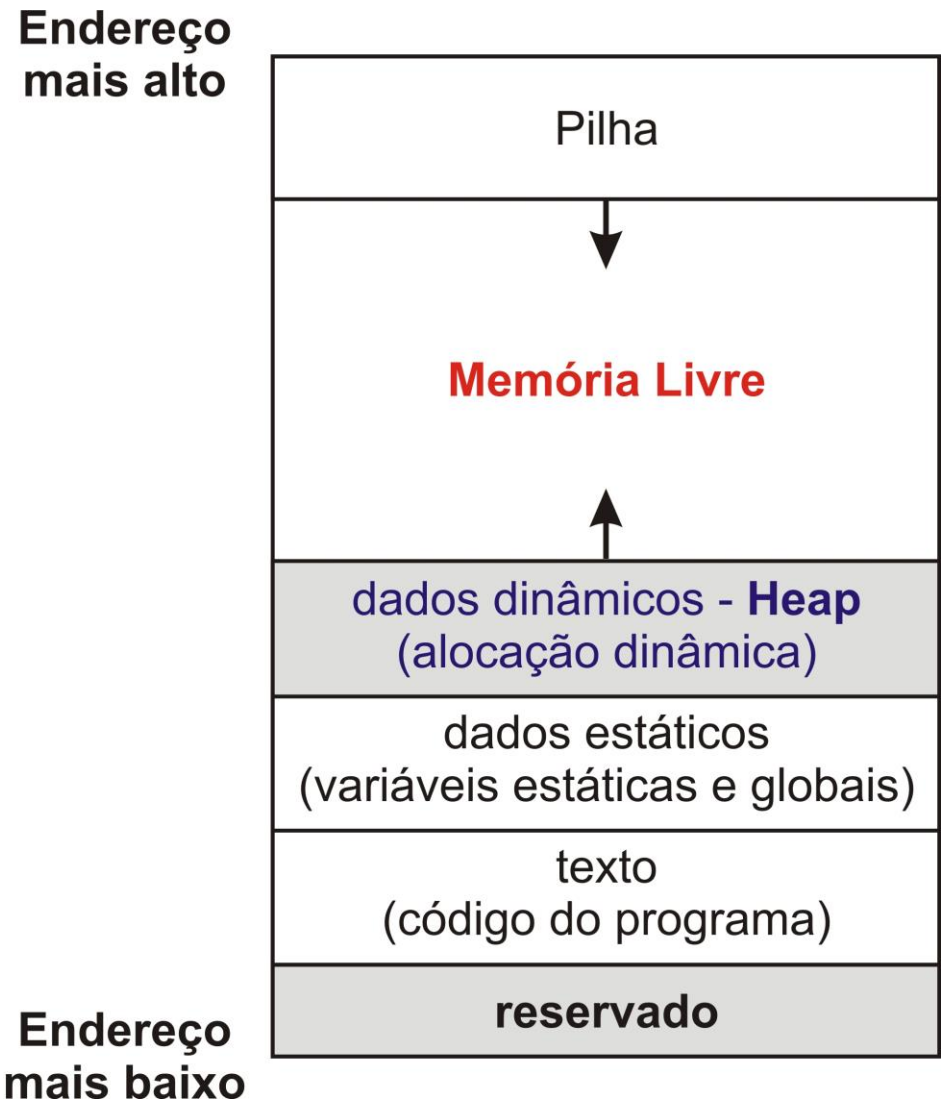
Memória RAM

A RAM é organizada em diferentes áreas de armazenamento para o correto funcionamento dos programas.

Cada área tem um papel essencial na execução e no isolamento dos processos.

Cada programa em execução possui seu próprio espaço de memória dividido em seções.

A estrutura ao lado ilustra essa divisão.



Alocação Estática

O espaço para as variáveis é reservado e liberado automaticamente pelo compilador.

```
#include <stdio.h>
	int main(){
	int a;
	int b[20]; // ALOCAÇÃO ESTÁTICA
	// CONTINUA
	return 0;
}
```

Ponteiros

Variáveis alocadas dinamicamente são chamadas de ponteiros ou apontadores (*pointers*), pois armazenam o endereço de memória de uma variável.

Heap

A memória alocada dinamicamente faz parte de uma área da memória chamada **heap**.

Basicamente, o programa aloca e desaloca porções de memória do heap durante a execução do programa.

Liberação de Memória

A memória deve ser liberada após o término de seu uso. Este trabalho deve ser feito por quem fez a alocação:

- Alocação Estática: compilador.
- Alocação Dinâmica: programador.

Alocação Dinâmica

O espaço para as variáveis é reservado e liberado dinamicamente pelo programador.

- sizeof(int) -> quantidade de bytes

```
#include <stdio.h>
#include <stdlib.h>
int main(){
	int *a = (int *)malloc(sizeof(int));
	// CONTINUA
}
```

Observações

No comando:

```
int *a = (int *)malloc(sizeof(int));
```

o (int *) é usado para fazer um cast (ou conversão) do tipo void * retornado por malloc para int *.

Esse cast pode ser adaptado para outros tipos de dados, como float *, double * e char *, conforme necessário.

Observações

Em C, a função malloc retorna um ponteiro genérico (void *), que pode apontar para qualquer tipo de dado, mas não possui um tipo específico.

Portanto, para associá-lo a um ponteiro de tipo específico, como int *, é necessário fazer esse cast.

Esse cast deixa claro para o compilador que o ponteiro retornado por malloc deve ser tratado como um ponteiro para determinado tipo de dado.

Alocação Dinâmica

A área de alocação dinâmica, também chamada heap, consiste em toda memória disponível que não foi usada para outro propósito.

A linguagem C oferece um conjunto de funções que permitem a alocação ou a liberação dinâmica de memória: malloc(), calloc(), realloc() e free( ).

As funções estão disponíveis na biblioteca <stdlib.h>.

sizeof

Exemplo de alocação dinâmica

```
int *a = (int *)malloc(sizeof(int));
```

- Em geral, um int ocupa 4 bytes de memória, mas isso pode variar.

O uso do **sizeof**

Em C, o tamanho de int e float pode variar conforme o sistema e o compilador.

Tipos Inteiros

- sizeof(int) — Tamanho padrão (geralmente 4 bytes)
- sizeof(short int) — short (geralmente 2 bytes)
- sizeof(long int) — long (4 ou 8 bytes)
- sizeof(long long int) — long long (normalmente 8 bytes)

O uso do **sizeof**

Tipos Ponto Flutuante:

- sizeof(float) — float (geralmente 4 bytes)
- sizeof(double) — double (geralmente 8 bytes)
- sizeof(long double) — long double (10, 12 ou 16 bytes)

O operador **sizeof** ajuda a verificar o tamanho exato para uso eficiente de memória.

malloc

malloc

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

- Ela retorna um ponteiro do tipo void que pode ser convertido em um ponteiro de qualquer tipo

- A posição alocada conterá “lixo”

Exemplo - malloc

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

int main(){
      int *ptr;
      int n = 5;
      ptr = (int *)malloc(n * sizeof(int));
      // Continua...
}
```

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

Exemplo - malloc

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
        int *pi;
        pi = (int *)malloc(sizeof(int));
        printf("Digite um número: ");
        scanf("%d", pi);
        printf("\nVocê digitou o número: %d", *pi);
        return 0;
}
```

Saída:

Digite um número: 56

Exemplo - malloc

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
        int *pi;
        pi = (int *)malloc(sizeof(int));
        printf("Digite um número: ");
        scanf("%d", pi);
        printf("\nVocê digitou o número: %d", *pi);
        return 0;
}
```

Saída:

Digite um número: 56

Você digitou o número 56

free

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

Exemplo - malloc

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
        int *pi;
        pi = (int *)malloc(sizeof(int));
        printf("Digite um número: ");
        scanf("%d", pi);
        printf("\nVocê digitou o número: %d", *pi);
        free(pi);
        printf("\nVocê digitou o número: %d", *pi);
        return 0;
}
```

Saída:

Digite um número: 56

Exemplo - malloc

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
        int *pi;
        pi = (int *)malloc(sizeof(int));
        printf("Digite um número: ");
        scanf("%d", pi);
        printf("\nVocê digitou o número: %d", *pi);
        free(pi);
        printf("\nVocê digitou o número: %d", *pi);
        return 0;
}
```

Saída:

Digite um número: 56

Você digitou o número: 56

Você digitou o número: 0

calloc

calloc

- A função calloc, ou “alocação contígua”, é usada para alocar dinamicamente um bloco de memória com múltiplos elementos.

- Ao contrário de malloc, calloc inicializa todas as posições com o valor zero.

- Retorna um ponteiro do tipo void *, que pode ser convertido para qualquer tipo de ponteiro.

Exemplo - calloc

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

int main(){
    int *ptr;
    int n = 5;
    ptr = (int *)calloc(n, sizeof(int));
    // Aloca espaço para 5 inteiros
    // Continua...
}
```

- Serão alocados 20 bytes para 5 inteiros de 4 bytes.
- Cada posição será inicializada como 0.
- Se não houver espaço disponível, calloc retorna NULL.

Exemplo - calloc

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

int main() {
	int *pi;
	pi = (int *)calloc(1, sizeof(int));
	printf("Digite um número: ");
	scanf("%d", pi);
	printf("\nVocê digitou o número: %d", *pi);
	free(pi);
	printf("\nVocê digitou o número: %d", *pi);
	return 0;
}
```

Saída:

Digite um número: 56

Exemplo - calloc

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

int main() {
	int *pi;
	pi = (int *)calloc(1, sizeof(int));
	printf("Digite um número: ");
	scanf("%d", pi);
	printf("\nVocê digitou o número: %d", *pi);
	free(pi);
	printf("\nVocê digitou o número: %d", *pi);
	return 0;
}
```

Saída:

Digite um número: 56

Você digitou o número: 56

Você digitou o número: 0

Demonstração da Inicialização com calloc

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

int main() {
	int *array;
	int n = 5;
	array = (int *)calloc(n, sizeof(int));
	for (int i = 0; i < n; i++) {
		printf("Valor em array[%d]: %d\n", i, array[i]);
	}
	free(array);
	return 0;
}
```

Demonstração da Inicialização com calloc

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

int main() {
	int *array;
	int n = 5;
	array = (int *)calloc(n, sizeof(int));
	for (int i = 0; i < n; i++) {
		printf("Valor em array[%d]: %d\n", i, array[i]);
	}
	free(array);
	return 0;
}
```

```
Valor em array[0]: 0
Valor em array[1]: 0
Valor em array[2]: 0
Valor em array[3]: 0
Valor em array[4]: 0
```

realloc

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.

Exemplo - realloc

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
	int *pi;
	pi = (int *)malloc(sizeof(int));
	pi = realloc(pi, 5 * sizeof(int));
	printf("Digite um número: ");
	scanf("%d", pi);
	printf("\nVocê digitou o número %d:", *pi);
	free(pi);
	printf("\nVocê digitou o número %d:", *pi);
	return 0;
}
```

Saída:

Digite um número: 34

Exemplo - realloc

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
	int *pi;
	pi = (int *)malloc(sizeof(int));
	pi = realloc(pi, 5 * sizeof(int));
	printf("Digite um número: ");
	scanf("%d", pi);
	printf("\nVocê digitou o número %d:", *pi);
	free(pi);
	printf("\nVocê digitou o número %d:", *pi);
	return 0;
}
```

Saída:

Digite um número: 34

Você digitou o número: 34

Você digitou o número: 0

Exemplo a seguir

- No exemplo a seguir, a função realloc é usada para redimensionar um array dinâmico.

- Este exemplo mostra como realloc preserva o conteúdo original ao expandir ou reduzir o tamanho do array.

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

int main() {
      int *arr = (int *)malloc(5 * sizeof(int));
      for (int i = 0; i < 5; i++) {
            arr[i] = i + 1;
      }

// Expansão do array para 8 elementos
      arr = (int *)realloc(arr, 8 * sizeof(int));
      for (int i = 5; i < 8; i++) {
            arr[i] = (i + 1) * 10; // Novos valores
      }
```

```c
    // Redução do array para 3 elementos
    arr = (int *)realloc(arr, 3 * sizeof(int));

    // Imprime os elementos
    for (int i = 0; i < 3; i++) {
        printf("%d ", arr[i]);
    }

    free(arr);
    return 0;
}
```

Exemplo - free

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
        int *pi;
        pi = (int *)malloc(sizeof(int));
        pi = realloc(pi, 5 * sizeof(int));
        printf("Digite um número: ");
        scanf("%d", pi);
        printf("\nVocê digitou o número %d:", *pi);
        free(pi);
        printf("\nVocê digitou o número %d:", *pi);
        return 0;
}
```

Saída:

Digite um número: 67

Exemplo - free

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
        int *pi;
        pi = (int *)malloc(sizeof(int));
        pi = realloc(pi, 5 * sizeof(int));
        printf("Digite um número: ");
        scanf("%d", pi);
        printf("\nVocê digitou o número %d:", *pi);
        free(pi);
        printf("\nVocê digitou o número %d:", *pi);
        return 0;
}
```

Saída:

Digite um número: 67

Você digitou o número: 67

Você digitou o número: 0

Resumo das funções de alocação dinâmica de memória

Função	Propósito	Sintaxe	Comportamento
malloc	Alocar um bloco de memória.	(tipo *) malloc(tamanho)	Memória não inicializada (contém 'lixo').
calloc	Alocar múltiplos blocos de memória.	(tipo *) calloc(num, tamanho)	Memória inicializada com zero.
realloc	Redimensionar um bloco de memória.	(tipo *) realloc(ptr, novo_tamanho)	Preserva o conteúdo existente, expande com 'lixo'.
free	Liberar memória alocada.	free(ptr)	Libera a memória, não define o ponteiro como NULL.

Erros comuns

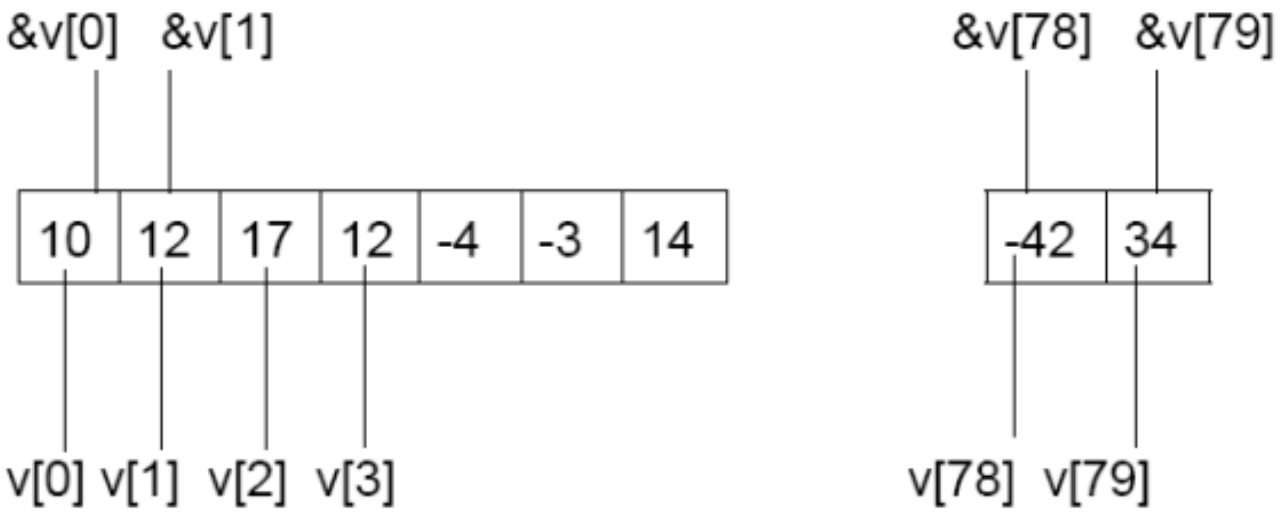
- Esquecer de desalocar memória. A memória será desalocada apenas no encerramento do programa, o que pode ser um grande problema em loops. Ocasiona “desperdício de memória”, que pode causar falha do sistema.

- Tentar acessar o conteúdo da variável depois de desalocá-la.

Vetores e ponteiros

Vetores

- Estruturas indexadas que armazenam dados do mesmo tipo

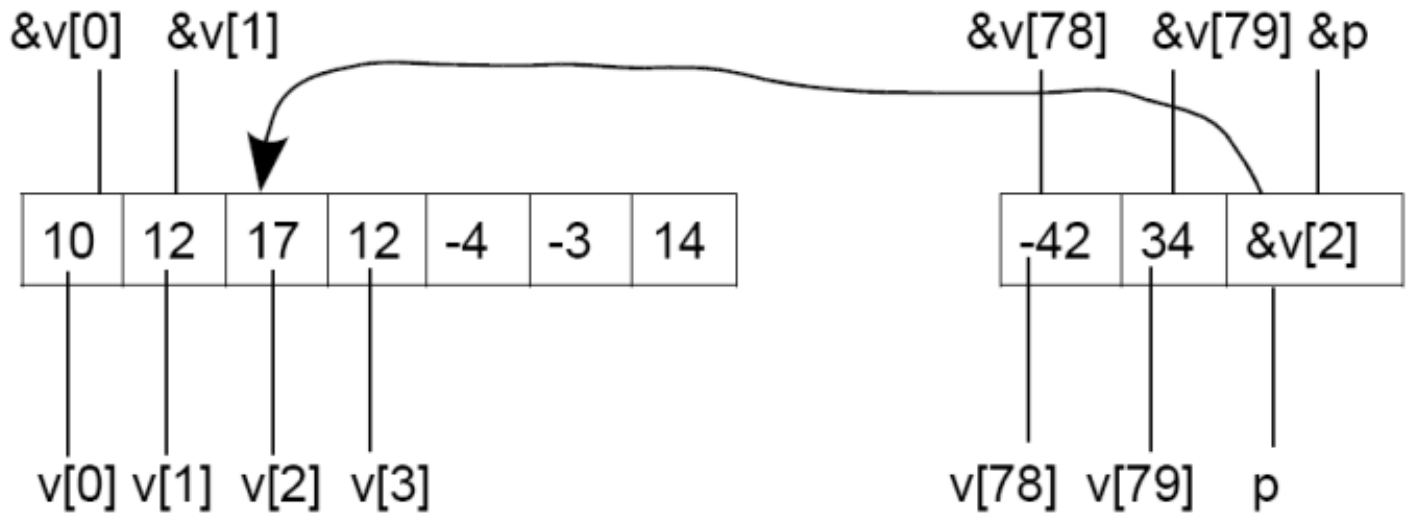


Vetores e ponteiros

- Facilita a manipulação de vetores:

```
int v[80];
int *p;

p = &v[2];
```

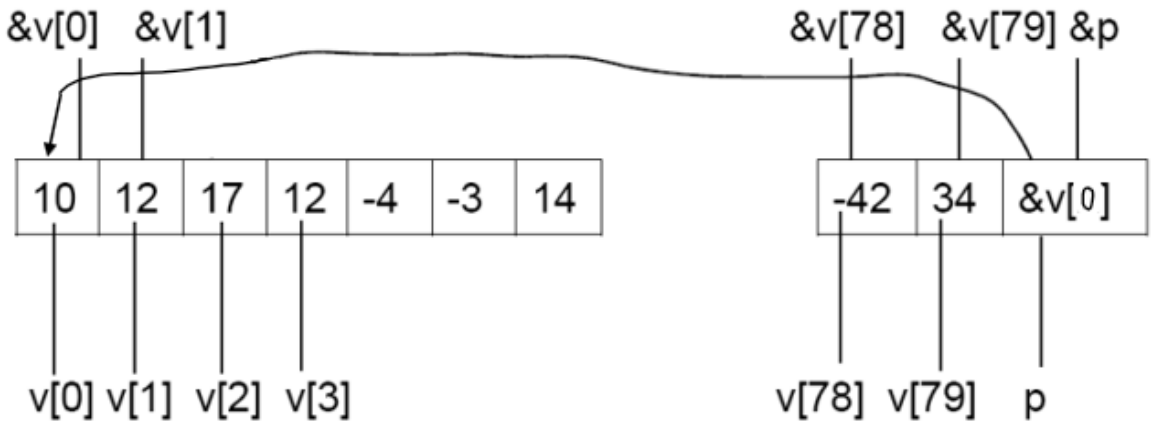


Vetores e ponteiros

- Aqui podemos usar também uma sintaxe especial, para fazer o ponteiro apontar para o início do vetor

```
int v[80];
int *p;

p = v;
```



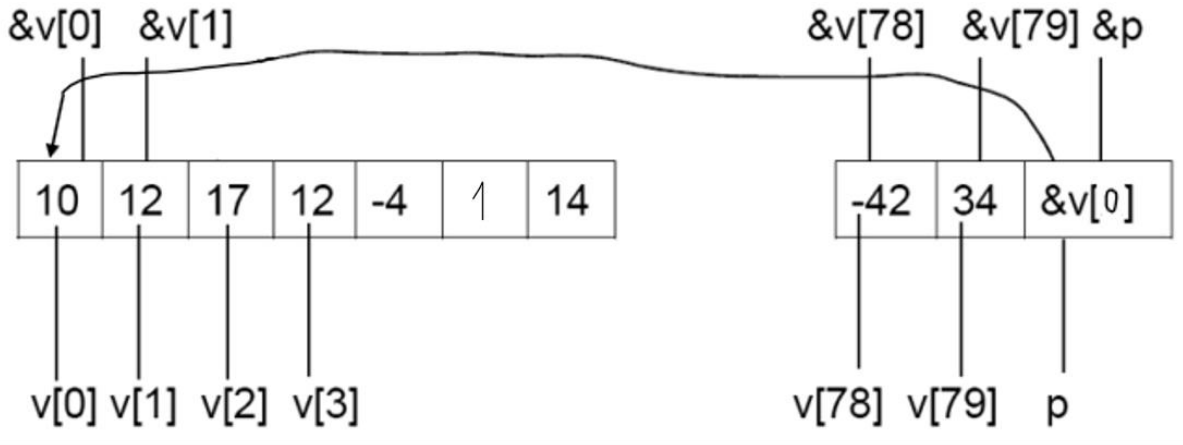
- Equivalente à p = &v[0];

Vetores e ponteiros

- Aqui podemos usar também uma sintaxe especial, para fazer o ponteiro apontar para o início do vetor

```
int v[80];
int *p;

p = v;
p[5] = 1;
```



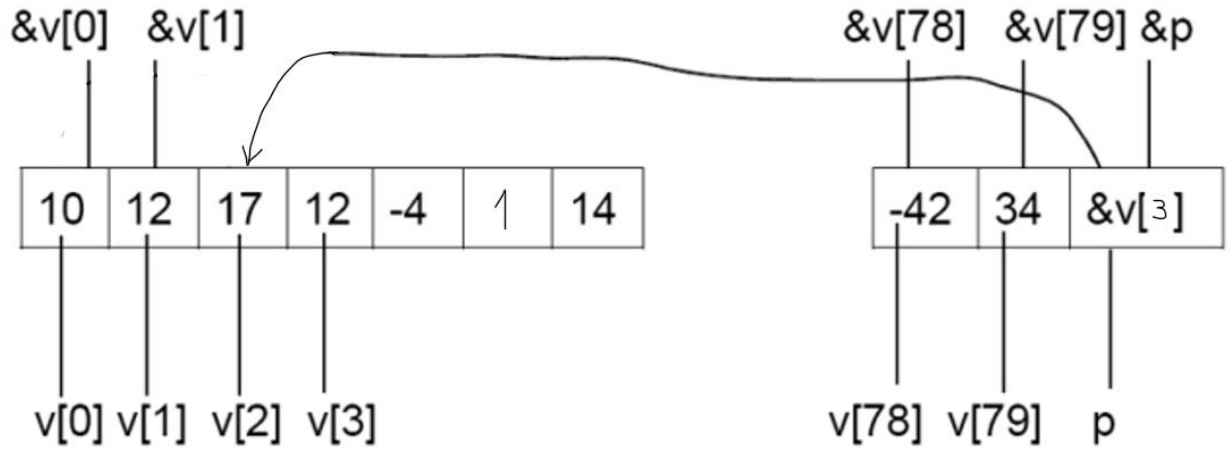
- Equivalente à v[5] = 1;

Vetores e ponteiros

- Aqui podemos usar também uma sintaxe especial, para fazer o ponteiro apontar para o início do vetor

```
int v[80];
int *p;

p = &v[3];
```



- Então, p[1] é o elemento v[4], p[2] o v[5] e assim por diante..

Operações

```
int *p, *q, n, v[50];
float *x, y[20];
```

- Se p = &v[4] ,
  - (p + 1) é &v[5] ; (p + 2) é &v[6] e assim por diante
  - *p é v[4], portanto se v[4] = = 3, *p = = 3.
  - *(p+1) é v[5] ..
- Se x = &y[3]
  - (x+1) = = &y[4].. (x+i) = = &y[3+1]

Operações

```
int *p, *q, n, v[50];
float *x, y[20];
```

- Somar ou subtrair um inteiro de um ponteiro

```
p = &v[22]; q = &v[30];
p = p - 4; q++;
*(p+2) = 3; *q = 4;
```

Operações

```
int *p, *q, n, v[50];
float *x, y[20];
```

- Somar ou subtrair um inteiro de um ponteiro

```
p = &v[22]; q = &v[30];
p = p - 4; q++;
*(p+2) = 3; *q = 4;
```


v[20] = 3
v[31] = 4

Vetores como parâmetros

```
# include <math.h>
float modulo (float v[], int n) {
  int i;
  float r = 0;
  for (i=0; i<n; i++) {
    r = r + v[i]*v[i];
  }
  r = sqrt (r);
  return r;
}
```

Vetores como parâmetros

```c
# include <math.h>
float modulo (float v[], int n) {
  int i;
  float r = 0;
  for (i=0; i<n; i++) {
    r = r + v[i]*v[i];
  }
  r = sqrt (r);
  return r;
}
```

```c
float modulo (float *p , int n) {
  int i;
  float r = 0;
  for (i=0; i<n; i++) {
    r = r + p[i]*p[i];
  }
  r = sqrt (r);
  return r;
}
```

float v[] é a mesma coisa que *p → v é um ponteiro!

Vetores como parâmetros

O parâmetro v da função modulo aponta para a variável x[0] da função main.

Então v[i] na função modulo é exatamente x[i] da função main.

v aponta para x[0]. Então v[i] é x[i]

```
1   # include <stdio.h>
2   # include <math.h>
3
4
5   float modulo (float v[], int n) {
6     int i;
7     float r = 0;
8     for (i=0; i<n; i++) {
9       r = r + v[i]*v[i];
10    }
11    r = sqrt (r);
12    return r;
13  }
14
15  int main () {
16    float x[100], comprimento;
17    int m;
18
19    m = 3;
20    x[0] = 2; x[1] = -3, x[2] = 4;
21
22
23    comprimento =  modulo (x, m);
24
25    printf ("Comprimento = %f\n", comprimento);
26
27    return 0;
28  }
```

Vetores como parâmetros

O parâmetro v da função modulo aponta para a variável x[0] da função main.

Então v[i] na função modulo é exatamente x[i] da função main.

u aponta para v[0].

```
1   # define MAX 200
2
3
4   float f (float u[]) {
5     float s;
6     /* declaração da função f */
7     ...
8     u[i] = 4;
9     ...
10    return s;
11  }
12
13  int main () {
14    float a, v[MAX]; /* declaração da variável a e vetor v */
15    ...
16    /* outras coisas do programa */
17
18
19    a = f (v);   /* observe que o vetor é passado apenas pelo nome */
20
21    ...
22
23    return 0;
24  }
```

Problema

Faça uma função que recebe dois vetores de tamanho n e retorna o seu produto escalar.

O protótipo dessa função seria:

```
float ProdutoEscalar (float u[], float v[], int n);
```

Problema

Faça uma função que recebe dois vetores de tamanho n e retorna o seu produto escalar.

O protótipo dessa função seria:

```
float ProdutoEscalar (float u[], float v[], int n);
```

A função recebe como parâmetros os vetores u e v, e um inteiro n. Uma possível solução para esta função seria:

```
float ProdutoEscalar (float u[], float v[], int n) {
  int i;
  float res = 0;
  for (i=0; i<n; i++)
  res = res + u[i] * v[i];
  return res;
}
```

Passagem de Parâmetros

- Em C, por padrão, o que existe é a passagem de parâmetro por valor, ou seja, é sempre criada uma variável como uma cópia.

- Logo, deve-se implementar a passagem por referência utilizando-se de ponteiros.

- Cuidado com a alocação de memória!

Passagem de Parâmetros – Valor vs. Referência

```
void SomaUm(int x, int *y) {
	x = x + 1;
	*y = (*y) + 1;
	printf("Funcao SomaUm: %d %d\n", x, *y);
}

int main() {
	int a = 0, b = 0;
	SomaUm(a, &b);
	printf("Programa principal: %d %d\n", a, b);
	return 0;
}
```

Passagem de Parâmetros – Valor vs. Referência

```c
void SomaUm(int x, int *y) {
	x = x + 1;
	*y = (*y) + 1;
	printf("Funcao SomaUm: %d %d\n", x, *y);
}

int main() {
	int a = 0, b = 0;
	SomaUm(a, &b);
	printf("Programa principal: %d %d\n", a, b);
	return 0;
}
```

```
Funcao SomaUm: 1 1
Programa principal: 0 1
```

Exemplo 01

Faça um programa que leia um valor *n*, crie dinamicamente um vetor de *n* elementos inteiros e passe esse vetor para uma função que deverá preenchê-lo com valores digitados pelo usuário. A leitura do vetor deverá ser feita via scanf. O programa deve mostrar o vetor preenchido.

Importante: não se esqueça de liberar a memória alocada para o vetor.

Exemplo 02

Escreva uma função que receba como parâmetro um vetor de inteiros (passagem por referência) e o tamanho do vetor (passagem por valor). A função deve retornar o maior elemento do vetor.

a) Escreva uma solução iterativa usando os índices do vetor.

b) Escreva uma solução iterativa usando o ponteiro do vetor.

Exemplo 02

Escreva uma função que receba como parâmetro um vetor de inteiros (passagem por referência) e o tamanho do vetor (passagem por valor). A função deve retornar o maior elemento do vetor.

a) Escreva uma solução iterativa usando os índices do vetor.

b) Escreva uma solução iterativa usando o ponteiro do vetor.

c) Escreva uma solução recursiva.

Exemplo 03

Problema de avaliação de um polinômio: escreva uma função que receba um vetor com os coeficientes $(a_0, a_1, \ldots, a_n)$, receba o valor de x e retorne o valor do polinônio para um valor n recebido.

$$P(n) = a_0 + a_1 x^1 + a_2 x^2 + \ \ldots + a_n x^n$$

a) Solução iterativa.
b) Solução recursiva.

Exmplo 04

Escreva uma função em C que receba dois vetores como parâmetros e verifica se o segundo vetor ocorre dentro do primeiro.