Dominando Estruturas de Dados 1

Vetores e Alocação Estática e Dinâmica

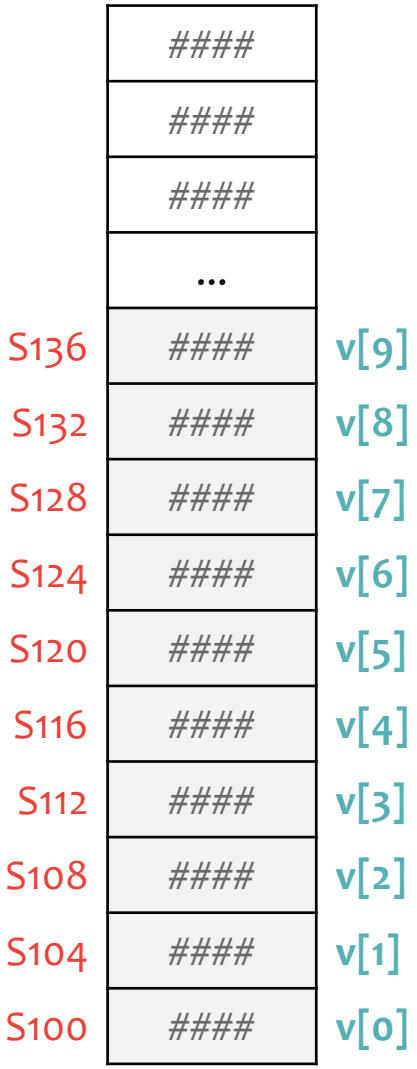
Prof. Samuel Martins (Samuka)
@xavecoding @hisamuka

xavecoding

Vetores

A forma mais simples de estruturarmos uma **lista de elementos** é por meio de **vetores/array**:

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



- Um **vetor de inteiros** dimensionado com **10** elementos;
- Reservamos um espaço de **memória contínuo** para armazenar **10** valores **inteiros**;
- Se cada **int** ocupa **4 bytes**, a declaração reserva um espaço de memória de **40 bytes**;

Vetores

- O acesso a cada elemento do vetor é feito através de uma **indexação** da variável **v**;
- Em C, a indexação de um vetor varia de **0 (zero)** a **n-1**, onde **n** representa a **dimensão do vetor (número de elementos)**.

- **v[0]** = acessa o primeiro elemento de **v**;
- **v[1]** = acessa o segundo elemento de **v**;
- ...
- **Cuidado: v[10] invade outra região de memória memória**

Vetores

- O acesso a cada elemento do vetor é feito através de uma **indexação** da variável **v**;
- Em C, a indexação de um vetor varia de 0 (zero) [illegible] enta a **dimensão do vetor (número**

```c
#include <stdio.h>

int main() {
    int v[5] = {0, 1, 2, 3, 4};

    for (int i = 0; i < 5; i++) {
        printf("&v[%d] = %p, v[%d] = %d\n", i, &v[i], i, v[i]);
    }

    printf("&v[6] = %p, v[6] = %d\n", &v[6], v[6]);

    return 0;
}
```

```
&v[0] = 0x7ffee290a660, v[0] = 0
&v[1] = 0x7ffee290a664, v[1] = 1
&v[2] = 0x7ffee290a668, v[2] = 2
&v[3] = 0x7ffee290a66c, v[3] = 3
&v[4] = 0x7ffee290a670, v[4] = 4
&v[6] = 0x7ffee290a678, v[6] = 231669951
```

Lixo de memória

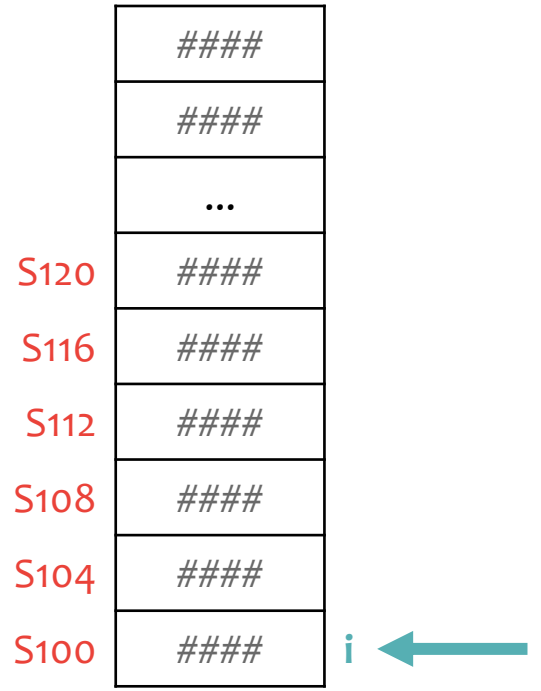
Vetores: Um exemplo de Leitura

```c
int i;
int v[5];

for (i = 0; i < 5; i++)
    scanf("%d", &v[i]);
```

Vetores: Um exemplo de Leitura

```
int i;
int v[5];

for (i = 0; i < 5; i++)
    scanf("%d", &v[i]);
```

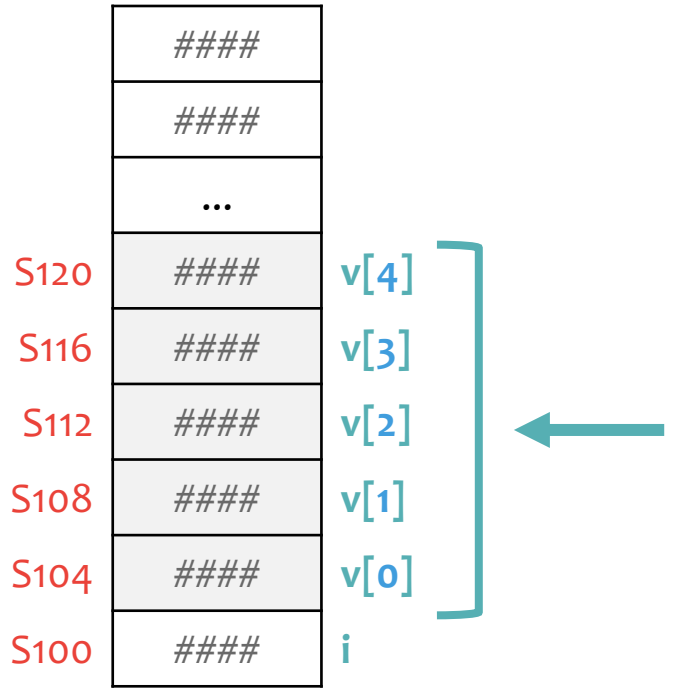
i = ####;



Vetores: Um exemplo de Leitura

```c
int i;
int v[5];

for (i = 0; i < 5; i++)
    scanf("%d", &v[i]);
```

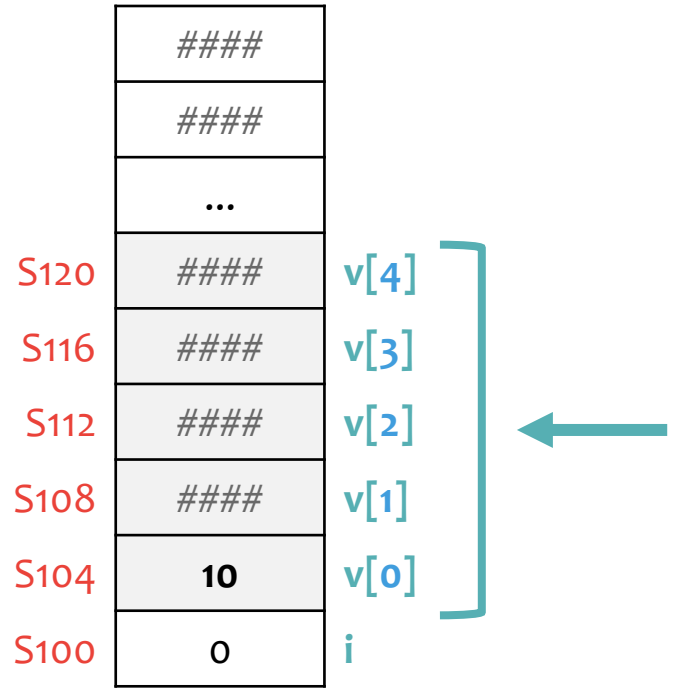
i = ####;
&v[0] = S104; v[0] = *(S104) = ####;
&v[1] = S108; v[1] = *(S108) = ####;
&v[2] = S112; v[2] = *(S112) = ####;
&v[3] = S116 v[3] = *(S116) = ####;
&v[4] = S120; v[4] = ####;



Vetores: Um exemplo de Leitura

scanf – recebe um endereço e atribui o valor digitado ao seu conteúdo

```
int i;
int v[5];

for (i = 0; i < 5; i++)
    scanf("%d", &v[i]);
```



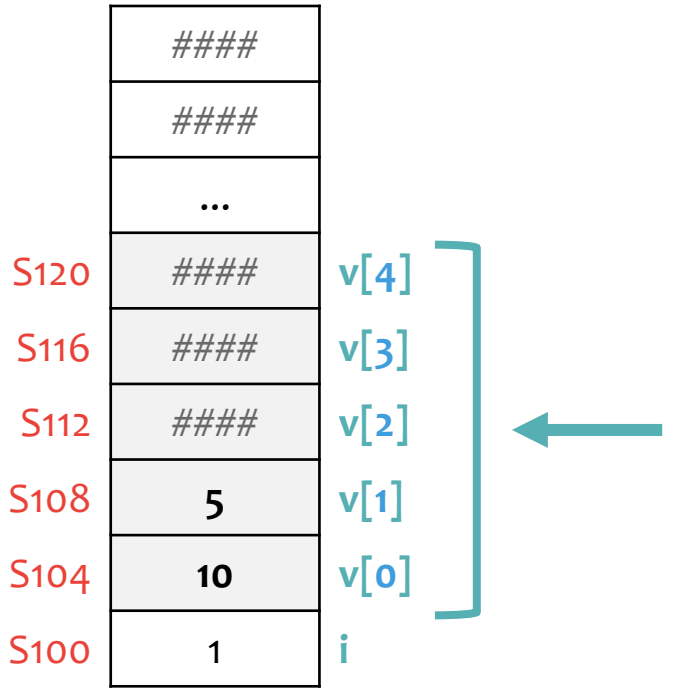
i = 0;
&v[0] = S104; v[0] = *(S104) = 10;
&v[1] = S108; v[1] = *(S108) = ####;
&v[2] = S112; v[2] = *(S112) = ####;
&v[3] = S116 v[3] = *(S116) = ####;
&v[4] = S120; v[4] = ####;

Vetores: Um exemplo de Leitura

scanf – recebe um endereço e atribui o valor digitado ao seu conteúdo

```
int i;
int v[5];

for (i = 0; i < 5; i++)
    scanf("%d", &v[i]);
```

i = 1;
&v[0] = S104; v[0] = *(S104) = 10;
&v[1] = S108; v[1] = *(S108) = 5;
&v[2] = S112; v[2] = *(S112) = ####;
&v[3] = S116 v[3] = *(S116) = ####;
&v[4] = S120; v[4] = ####;

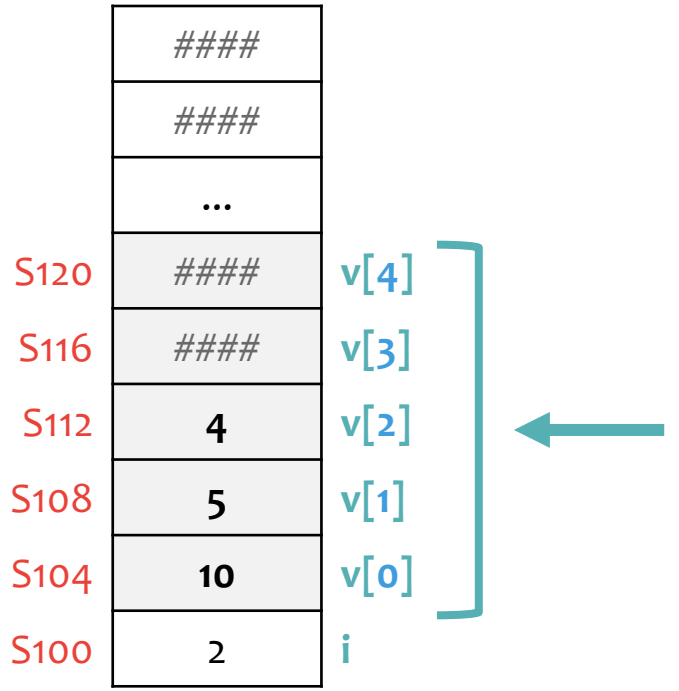


Vetores: Um exemplo de Leitura

scanf – recebe um endereço e atribui o valor digitado ao seu conteúdo

```
int i;
int v[5];

for (i = 0; i < 5; i++)
    scanf("%d", &v[i]);
```

i = 2;
&v[0] = S104; v[0] = *(S104) = 10;
&v[1] = S108; v[1] = *(S108) = 5;
&v[2] = S112; v[2] = *(S112) = 4;
&v[3] = S116 v[3] = *(S116) = ####;
&v[4] = S120; v[4] = ####;

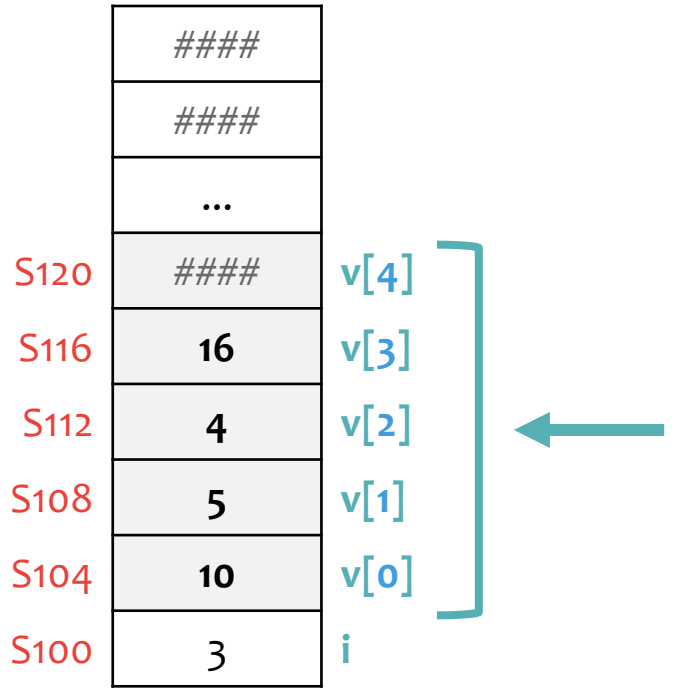


Vetores: Um exemplo de Leitura

scanf – recebe um endereço e atribui o valor digitado ao seu conteúdo

```
int i;
int v[5];

for (i = 0; i < 5; i++)
    scanf("%d", &v[i]);
```

i = 3;
&v[0] = S104; v[0] = *(S104) = 10;
&v[1] = S108; v[1] = *(S108) = 5;
&v[2] = S112; v[2] = *(S112) = 4;
&v[3] = S116 v[3] = *(S116) = 16;
&v[4] = S120; v[4] = ####;

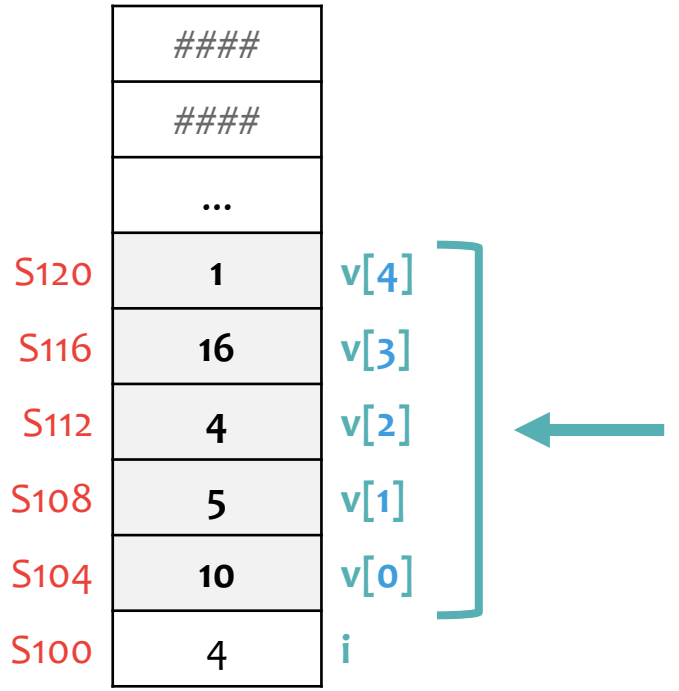


Vetores: Um exemplo de Leitura

scanf – recebe um endereço e atribui o valor digitado ao seu conteúdo

```
int i;
int v[5];

for (i = 0; i < 5; i++)
    scanf("%d", &v[i]);
```

i = 4;
&v[0] = S104; v[0] = *(S104) = 10;
&v[1] = S108; v[1] = *(S108) = 5;
&v[2] = S112; v[2] = *(S112) = 4;
&v[3] = S116 v[3] = *(S116) = 16;
&v[4] = S120; v[4] = *(S120) = 1;

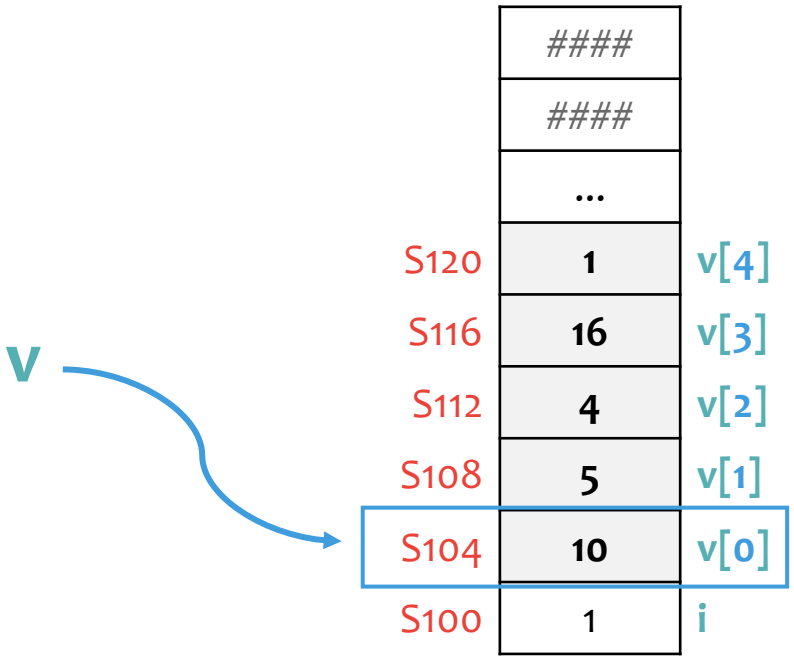


Aritmética de Ponteiros

Ao declararmos: `int v[5];`

- O símbolo **v** é uma constante que representa o **endereço inicial** do vetor.
- Logo, **sem indexação**, **v aponta (guarda a referência)** para o **primeiro elemento** do vetor.

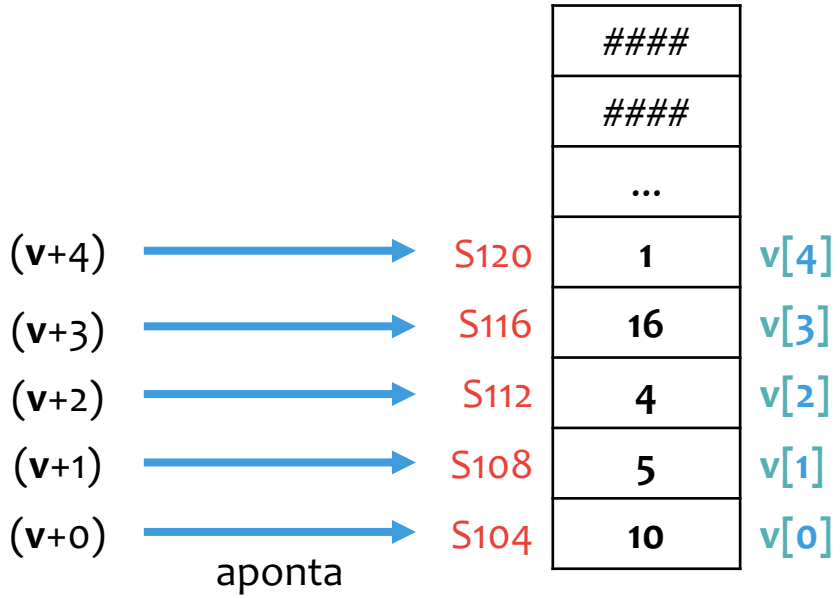
&v = v = &v[0]



Aritmética de Ponteiros

Logo:

-> **apontar** para uma variável **x** significa "**possui/armazena**" o endereço da variavel **x**

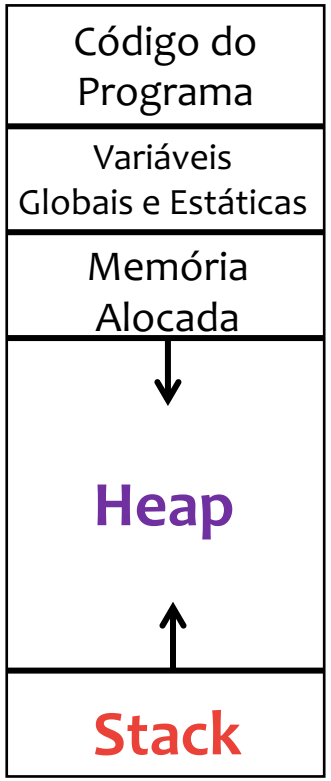


(v+0) = &v[0] = S104;
(v+1) = &v[1] = S108;
(v+2) = &v[2] = S112;
(v+3) = &v[3] = S116;
(v+4) = &v[4] = S120;

*(v+0) = *(S104) = v[0] = **10;**
*(v+1) = *(S108) = v[1] = **5;**
*(v+2) = *(S112) = v[2] = **4;**
*(v+3) = *(S116) = v[3] = **16;**
*(v+4) = *(S120) = v[4] = **1;**

Tipos de Alocações de Memória

Esquema de Memória



Alocação Estatíca

- O espaço para as variáveis é reservado no **início da execução**;
- Cada variável tem seu **endereço** fixado e a área de memória ocupada por ela se **mantém constante** durante **toda a execução**;
- São alocadas na **Stack** da Memória Ram;
- **Liberação** de memória feita **automaticamente** pelo **compilador**.

```
int a;
float b;
char c;
int a[10];
float *p;
```

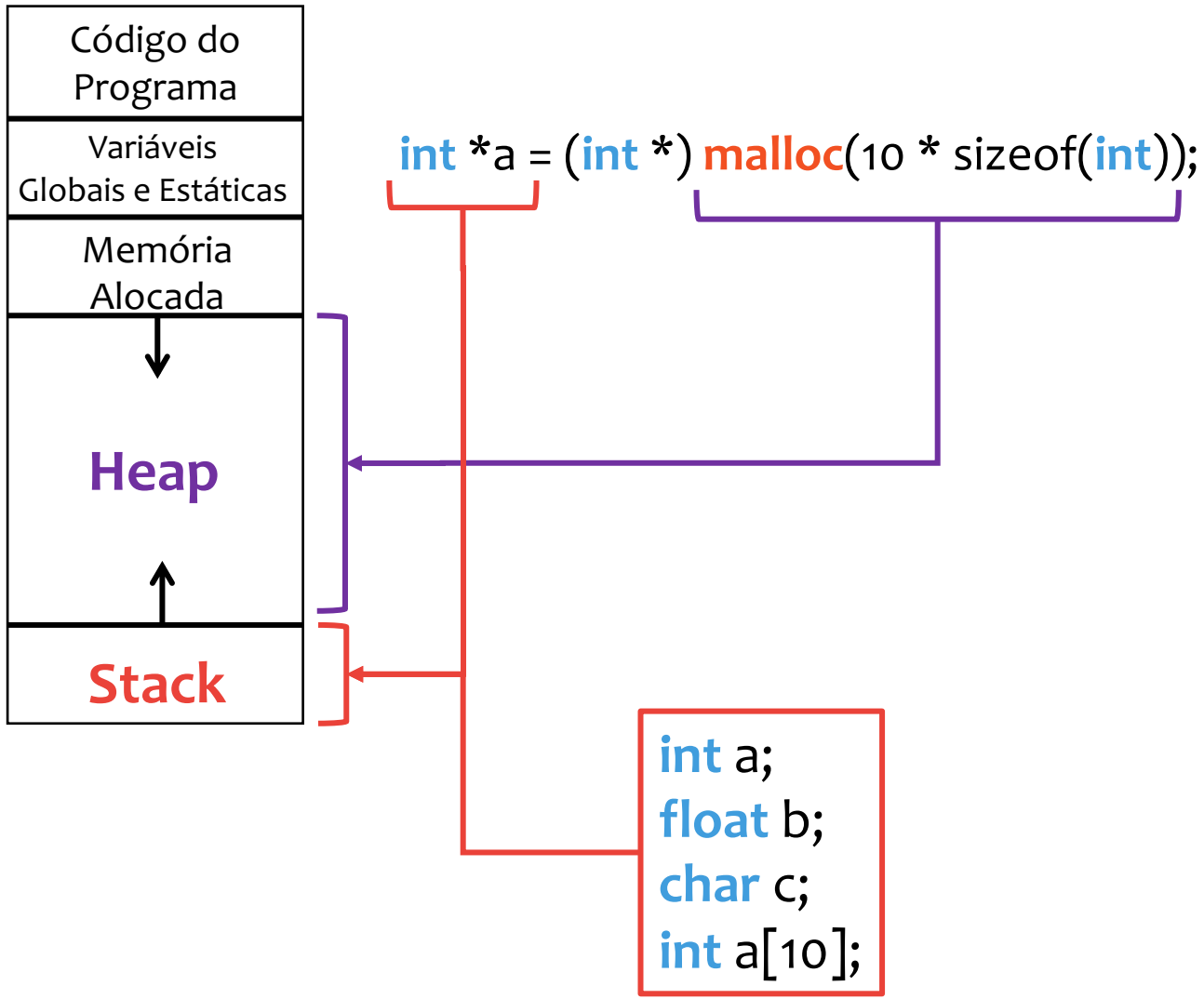
Alocação Dinâmica

- O espaço é alocado dinamicamente **durante a execução do programa**;
- Pode ser criada ou eliminada durante a **execução do programa**, ocupando espaço na memória **apenas** enquanto está sendo utilizada.
- São alocadas na **Heap (free store)** da Memória Ram;

- Liberação de memória feita **manualmente** pelo **programador** → **PERIGO!**

```
int *a = (int *) malloc(10 * sizeof(int));
float *b = (float *) calloc(5, sizeof(float));
free(a);
free(b);
```

Esquema de Memória



Alocação Dinâmica

Por que Usar?

- A **alocação dinâmica** é o processo que aloca memória em **tempo de execução**.
- Ela é utilizada quando não se sabe ao certo **quanto de memória será necessário** para o armazenamento dos elementos;
- Assim, o tamanho de memória necessário é determinado conforme necessidade;
- Dessa forma evita-se o desperdício de memória;

Além disso, **size(Heap) >> size(Stack)**.

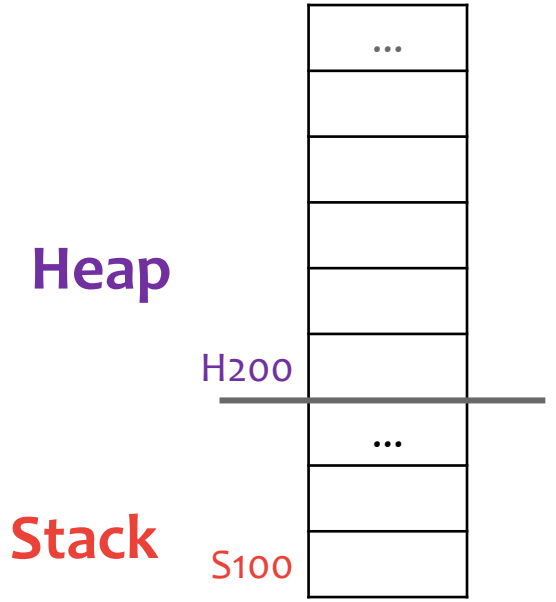
Alocação Dinâmica

malloc

- Aloca um **bloco de bytes consecutivos** na **memória heap** e devolve o endereço desse bloco.

```
tipo* v = (tipo*) malloc(n * sizeof(tipo));
          opcional
```

```
int* v = (int*) malloc(5 * sizeof(int));
```



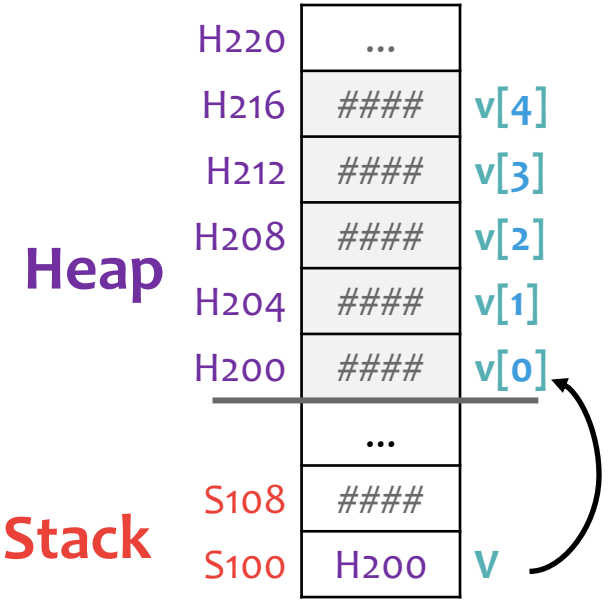
Alocação Dinâmica

malloc

- Aloca um **bloco de bytes consecutivos** na **memória heap** e devolve o endereço desse bloco.

```
tipo* v = (tipo*) malloc(n * sizeof(tipo));
          opcional
```

```
int* v = (int*) malloc(5 * sizeof(int));
```



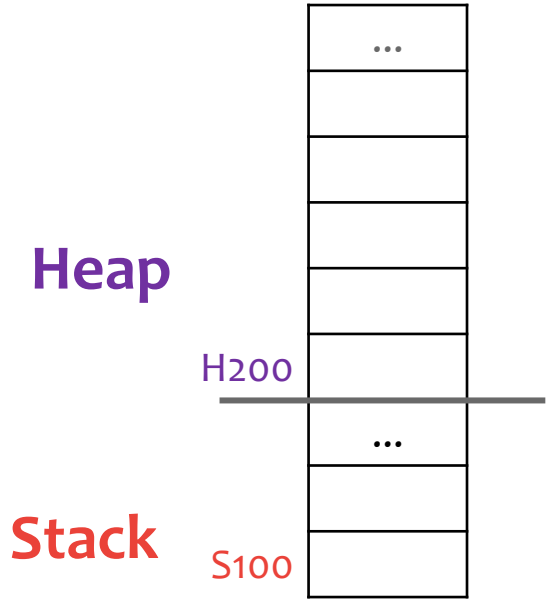
Alocação Dinâmica

calloc

- Aloca um **bloco de bytes consecutivos** na **memória heap** e inicializa todos os valores com **0** (**NULL** para ponteiros).

```
tipo* v = (tipo*) calloc(n, sizeof(tipo));
          opcional
```

```
int* v = (int*) calloc(5, sizeof(int));
```



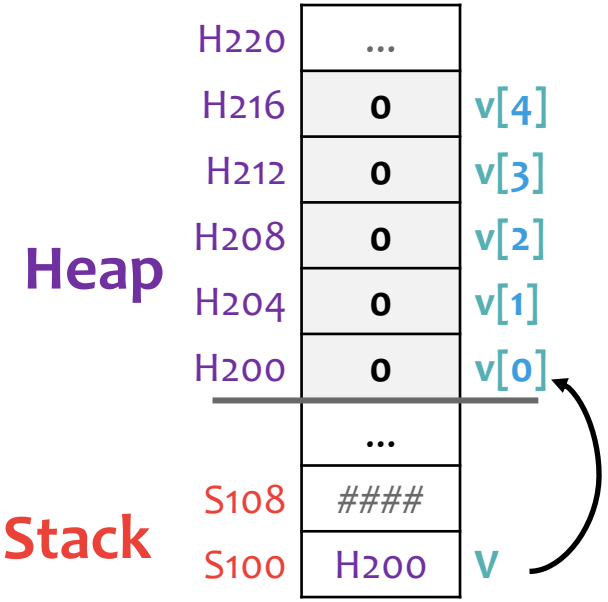
Alocação Dinâmica

calloc

- Aloca um **bloco de bytes consecutivos** na **memória heap** e inicializa todos os valores com **0** (**NULL** para ponteiros).

```
tipo* v = (tipo*) calloc(n, sizeof(tipo));
          (opcional)
```

```
int* v = (int*) calloc(5, sizeof(int));
```



Alocação Dinâmica

free

- Libera a porção de memória **Heap** alocada por **malloc** ou **calloc**, no qual é apontada por um ponteiro.

```
free(v);
```

BOA PRÁTICA DE PROGRAMAÇÃO:
Sempre após dar o **free** em um **ponteiro**, atribua **NULL**

```
free(v);
v = NULL;
```

Por quê?

Funções com Vetores Estáticos e Dinâmico

Let's code!

- Programa com vetores estáticos:
    - Crie uma função que recebe o ponteiro de um vetor e seu tamanho e imprima os elementos do vetor
    - Crie uma função que recebe o ponteiro de um vetor via colchetes [] e seu tamanho e imprima os elementos do vetor
    - Imprima os endereços de memória e valores do vetor na main e dentro de cada função;

- Programa com vetores dinâmicos:
    - Crie uma função que recebe o ponteiro de um vetor e seu tamanho e imprima os elementos do vetor
    - Crie uma função que recebe o ponteiro de um vetor via colchetes [] e seu tamanho e imprima os elementos do vetor
    - Imprima os endereços de memória e valores do vetor na main e dentro de cada função;

Exercícios

Ex 1) Simule a memória, usando **heap** e **stack**, para o seguinte trecho de código:

```c
int i, n = 5;
int* v;
v = (int*) malloc(n * sizeof(int));

for (i = 0; i < 5; i++) {
    v[i] = i;
}
```

Exercícios

Ex 2) Simule a memória, usando **heap** e **stack**, para o seguinte trecho de código:

```c
int v1[5] = {0, 1, 2, 3, 4};
int *v2, *p;
int i;

p = v1;
p[3] = p[4] = 10;

v2 = (int*) malloc(5 * sizeof(int));

for (i = 0; i < 5; i++) {
    v2[i] = v1[i];
}

free(v2);
v2 = NULL;
```

Exercícios

Ex 3) Qual o problema do trecho de código abaixo?

```
int v1[5] = {0, 1, 2, 3, 4};
int* v2;

v2 = (int*) malloc(5 * sizeof(int));
v2 = v1;

free(v2);
```

Exercícios

Ex 4) Qual o problema do trecho de código abaixo?

```c
int main() {
    int i;
    char* v;

    for (i = 0; i < 9999999; i++) {
        v = (char*) malloc(5000 * sizeof(char));
    }

    return 0;
}
```

Exercícios

Ex 5) Implemente e simule a memória para o que se pede abaixo:

1. Crie uma função que aloque um vetor de double e o retorne;
2. Crie uma função que aloque um vetor de double retornando-o por referência;
3. c) Crie uma função que desaloque um dado vetor, “setando-o” como NULL após a desalocação’
4. Crie uma função que faça a cópia de um vetor: faça a versão com retorno da função e com retorno por referência;
5. Crie uma função que calcule o mínimo e o máximo de um vetor de inteiros e retorne os valores em duas variáveis diferentes.

Dominando Estruturas de Dados 1

Vetores e Alocação Estática e Dinâmica

Prof. Samuel Martins (Samuka)
@xavecoding @hisamuka

xavecoding