CENTRO DE TECNOLOGIA - UFSM

Universidade Federal de Santa Maria
1960

Estruturas de Dados e Programação

UFSM00272

Professor Jonas Bulegon Gassen
jonas.gassen@ufsm.br

Arrays dinâmicos

Arrays dinâmicos

- Considere um vetor como esse:
  - [1, 3, 8, 9, 10, 15]

- Como podemos fazer uma função que complete os espaços do array?
- Por exemplo, entre o 1 e o 3, completar: 1, 2, 3

- A saída esperada é:
  - [1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

Arrays dinâmicos

- Antes de discutir em C, vamos fazer esse exemplo em Python.

Arrays dinâmicos

```
vet = [1, 3, 8, 9, 10, 15]
i = 0
while i < len(vet):
    if i < len(vet)-1:
        if vet[i+1] - vet[i] > 1:
            vet.insert(i+1, vet[i]+1)
    i += 1
print(vet)
```

Arrays dinâmicos

- Podemos fazer isso em C?
- Como?

Arrays dinâmicos

- Em C, nosso vetor terá um tamanho fixo, alocado para o mesmo.

- Se precisarmos de mais espaço, teremos que realocar o espaço reservado para o vetor.

- O que precisamos para esse fluxo?

Como ajustar?

- Controlar a quantidade máxima de elementos do nosso vetor;

- Controlar a quantidade atual de elementos do nosso vetor;

- Verificar se a quantidade atual já chegou na máxima e, nesse caso, realocar o vetor para um espaço maior.

Alocação dinâmica

- Usamos a função malloc (“memory allocation”)
- Está na biblioteca padrão de C “stdlib.h”
- Aloca dinamicamente um bloco de memória no heap em tempo de execução
- void *malloc(size_t size);
  - size é o número de bytes a ser alocado.
  - retorna um ponteiro void, que deve ser convertido para o tipo de ponteiro apropriado.
  - Se falhar, retorna NULL
- A memória alocada não é inicializada (pode conter lixo)
- Após o uso, deve-se liberar a memória com free, evitando vazamentos.

Memória

Tipo de variável	Onde fica?	Tempo de vida
Variáveis locais (automáticas)	**Stack**	Durante o escopo da função
Variáveis `malloc`/`calloc`	**Heap**	Até você dar `free()`

Forma de declaração	Onde fica?	Exemplo em C
`char s[] = "Olá";`	Stack	`s[0] = 'X';`
`char *s = malloc(...);`	Heap	`strcpy(s, "Olá"); s[0] = 'X';`

- A stack é rápida, mas pequena e limitada.
- A heap é maior, mas requer gerenciamento manual (alocar/liberar).

Alocação dinâmica

- Para um vetor de 5 elementos:
  - **int** *vetor = malloc(5 * sizeof(int));

- Generalizando, malloc(**N** * sizeof(int))

- Caso o seu dado seja de outro tipo:
  - **float** *vetor = malloc(5 * sizeof(int));

Alocação dinâmica

- Para realocar espaço, usamos a função realloc, ex:

```
int *vetor = malloc(5 * sizeof(int));
...
int *temp = realloc(vetor, 10 * sizeof(int));
```

Alocação dinâmica

- É indicado verificar se o malloc não retornou NULL, caso não tenha conseguido alocar a memória.

```
int *vetor = malloc(tamanho * sizeof(int));
if (vetor == NULL) {
    printf("Erro na alocação de memória\n");
    return 1;
                                                }
```

- Deve-se verificar isso para o realloc também.

Exercício

- Construir um array dinâmico em C
- Controlar tamanho máximo
- Controlar tamanho atual
- Verificar se há espaço disponível para inserir novo elemento
  - Se não houver, **reallocar** vetor com o dobro do espaço
- Permitir que o usuário selecione a posição da nova inserção
- Ao inserir elemento, movimentar os subsequentes uma posição

Solução apenas com main

CENTRO DE TECNOLOGIA - UFSM

Universidade Federal de Santa Maria
SEDES SAPIENTIÆ
1960

Solução 1

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    size_t capacidade = 2;          // capacidade inicial
    size_t tamanho   = 0;           // quantidade de itens armazenados
    int   *vetor     = malloc(capacidade * sizeof(int));
    if (!vetor) { printf("Erro"); return 1; }
    for (int v = 0; v < 6; ++v) {
        if (tamanho == capacidade) { /* se encheu, dobra-se a capacidade */
            capacidade *= 2;
            int *tmp = realloc(vetor, capacidade * sizeof(int));
            if (!tmp) { printf("Erro"); free(vetor); return 1; }
            vetor = tmp;
            printf("Aumentou tamanho para %lu\n", capacidade);
        }
        vetor[tamanho++] = v;
    }
    printf("Vetor final (tamanho = %zu, capacidade = %zu):\n", tamanho, capacidade);
    for (size_t i = 0; i < tamanho; ++i)
        printf("[%zu] = %d\n", i, vetor[i]);
    free(vetor);
    return 0;
}
```

Solução com funções

CENTRO DE TECNOLOGIA - UFSM

Universidade Federal de Santa Maria
SEDES SAPIENTIÆ
1960

Solução 2

- Quando utilizamos funções, temos que ter um pouco de cuidado com os ponteiros. Vejamos com exemplos:

```
void exemplo(int *ptr) {
    printf("Função ponteiro: %p\n", ptr);
    printf("Função endereço variável: %p\n", &ptr);
    ptr[2] = 10;
}
int main() {
    int *vetor = malloc(5 * sizeof(int));
    printf("%d\n", vetor[2]);
    exemplo(vetor);
    printf("main ponteiro: %p\n", vetor);
    printf("main endereço variável: %p\n", &vetor);
    printf("%d\n", vetor[2]);
    return 0;
}
```

Solução 2

- No código do slide anterior, enviamos o ponteiro para a função e, através do endereço do ponteiro, modificamos um valor no vetor.
- Usamos a referência e, portanto, a mudança afeta o main, visto que acessamos o mesmo ponto da memória.

- E se alterarmos, com realloc, o endereço do vetor?

Solução 2

- Realloc na função:

```
void exemplo(int *ptr) {
    printf("Função ponteiro: %p\n", ptr);
    printf("Função endereço variável: %p\n", &ptr);
    ptr[2] = 10;
    int *tmp = realloc(ptr, 10 * sizeof(int));
    if(tmp != NULL) {
        ptr = tmp;
    }
    printf("Função 2 ponteiro: %p\n", ptr);
    printf("Função 2 endereço variável: %p\n", &ptr);
}
```

- Assim, mudamos apenas para onde o ponteiro **local** da função está apontando, ou seja, o valor armazenado na variável ptr
- Essa mudança não reflete o ponteiro do main, que era nosso objetivo

Solução 2

- Temos que enviar o endereço do ponteiro (ponteiro para ponteiro)

```
void exemplo(int **ptr) {
    printf("Função ponteiro: %p\n", *ptr);
    printf("Função endereço variável: %p\n", &ptr);
    (*ptr)[2] = 10;
    int *tmp = realloc(*ptr, 10 * sizeof(int));
    if(tmp != NULL) {
        *ptr = tmp;
    }
    printf("Função 2 ponteiro: %p\n", *ptr);
    printf("Função 2 endereço variável: %p\n\n", &ptr);
}
```

- Assim, alteramos o endereço armazenado no ponteiro do main, atualizando aquele ponteiro quando o realloc tem sucesso.

Solução 2

- Para isso, na main devemos enviar o endereço do ponteiro, na chamada da função:

```
int main() {
    int *vetor = malloc(5 * sizeof(int));
    printf("%d\n", vetor[2]);
    exemplo(&vetor);
    printf("main ponteiro: %p\n", vetor);
    printf("main endereço variável: %p\n", &vetor);
    printf("%d\n", vetor[2]);
    return 0;
}
```

Solução 2

Com isso em mente, faça as funções de acordo.