Alocação de memória e Estruturas

Prof. Dr. Oscar Eduardo Anacona Mosquera

Universidade Federal de Mato Grosso
Faculdade de Engenharia
Campus Várzea Grande
Engenharia da Computação

21 de agosto de 2023

Conteúdo

1. Objetivos
2. Ponteiros
3. Alocação
   - Malloc
   - Calloc
   - Realloc
   - Memory leak

Conteúdo

1 Objetivos

2 Ponteiros

3 Alocação
- Malloc
- Calloc
- Realloc
- Memory leak

- **Compreensão dos Conceitos Básicos:**
  - Introduzir os conceitos de vetores unidimensionais e multidimensionais em C.
  - Explicar como os ponteiros podem ser utilizados para manipular e acessar elementos de vetores.
- **Alocação de Memória Dinâmica:**
  - Ensinar o funcionamento da alocação dinâmica de memória utilizando os comandos **malloc**, **calloc** e **free**.
- **Identificação e Prevenção de Memory Leaks:**
  - Explicar o conceito de ”memory leak”(vazamento de memória) e por que ele ocorre.
  - Ensinar técnicas para identificar e prevenir vazamentos de memória, destacando a importância da liberação correta da memória alocada dinamicamente.

Conteúdo

1 Objetivos

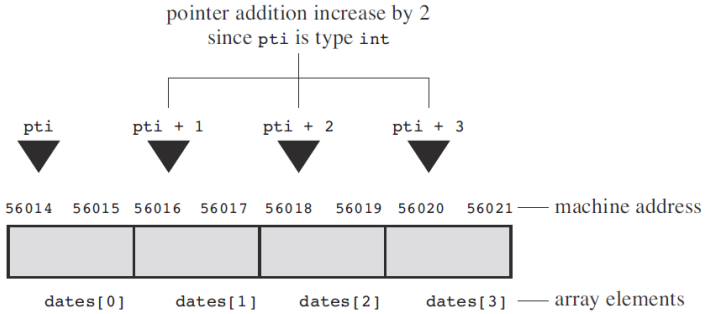
2 Ponteiros

3 Alocação
- Malloc
- Calloc
- Realloc
- Memory leak

Ponteiros e vetores

Vetores

Vetores são sequências de elementos de um tipo de dado, armazenados em posições contíguas de memória, distribuídas em um número predeterminado de dimensões.




```
int dates[y], *pti;
pti = dates; (or pti = & dates[0];)
```

pointer variable `pti` is assigned the address of the first element of the array `dates`

Referenciando os elementos dos vetores

Vetores

As referências aos elementos dos vetores são implementadas através da aritmética de ponteiros.

Exemplo: vetor 1-d

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

int main()
{
    int vet[5]={10,2,3,5,14};
    int *ptr_vet;

    ptr_vet=&vet;

    printf("%d\n",vet[1]);
    printf("%d\n",*(ptr_vet+1));

    return 0;
}
```

Referenciando os elementos dos vetores

Exemplo: vetor 2-d

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

int main()
{
    int vet[3][3]={{10,2,3},{5,14,-3},{9,6,4}};
    int *ptr_vet;

    int row=3,col=3,i,j;

    ptr_vet=&vet;
    i=1;
    j=2;

    printf("%d\n",vet[i][j]);

    printf("%d\n",*(ptr_vet + row*i + j));

    return 0;
}
```

Conteúdo

1 Objetivos

2 Ponteiros

3 Alocação
- Malloc
- Calloc
- Realloc
- Memory leak

Alocação de memória

Existem três tipos de alocação:

- **Estático:** a variável é alocada uma única vez, antes do início da execução do programa, e permanece alocada durante toda a execução.
- **Automático:** a variável é alocada sempre que a execução da programação inicia o bloco no qual ela é declarada, permanecendo alocada até que o bloco seja finalizado. O valor inicial é indeterminado mas sempre que a declaração da variável é executada, ela assume o valor da sua expressão de iniciação, se houver, ou um valor indeterminado, em caso contrário.
- **Por comando:** a alocação ocorre em decorrência da execução de comandos próprios de alocação de memória.

Alocação de memória

As funções de gerenciamento de memória permitem alocar, realocar e liberar espaços de memória, e são declaradas no cabeçalho **stdlib.h**.

- void *malloc(*size_t tam*):
  Aloca espaço de memória de tamanho igual a **tam bytes**. O conteúdo do espaço alocado é indeterminado.
- void *realloc(void *ptr,*size_t tam*):
  Desloca o espaço apontado por **ptr**, realocando seu conteúdo em um novo espaço de tamanho igual a **tam bytes**.
- void *calloc(unisgned int num, *size_t tam*):
  Aloca uma quantidade de memória igual a **tam bytes**.

free()

Comando necessário para liberar memória.

Vazamento de memória (Memory leak)

Memory leak

O vazamento de memória é caracterizado pela existência de espaço de memória alocado, mas que não pode ser acessado. Quando um espaço de memória é alocado pelas funções **malloc**, **calloc** e **realloc**, ele permanece alocado até o término do programa ou até que seja explicitamente desalocado.

Exemplos de alocação de memória

Exemplo: malloc

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

int main()
{
    int *mat;
    int lin=3,col=3,i,j;

    mat=(int *)malloc(lin*col*sizeof(int));

    *(mat + col*0 + 0)=2;

    for(i=0;i<lin;i++){
        for(j=0;j<col;j++){
            printf("%d\n",*(mat + col*i + j));
        }
    }
    return EXIT_SUCCESS;
}
```

Exemplos de alocação de memória

Exemplo: calloc

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

int main()
{
    int *mat;
    int lin=3,col=3,i,j;

    mat=(int *)calloc(lin*col,sizeof(int));

    *(mat + col*0 + 0)=2;

    for(i=0;i<lin;i++){
        for(j=0;j<col;j++){
            printf("%d\n",*(mat + col*i + j));
        }
    }
    return EXIT_SUCCESS;
}
```

Exemplos de alocação de memória

Exemplo: realloc

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

int main()
{
    int *mat,*vet;
    int lin=3,col=3,i,j;

    mat=(int *)calloc(lin*col,sizeof(int));

    *(mat + col*0 + 0)=2;

    vet=(int *)realloc(lin*col,sizeof(int));
    vet=mat;

    for(i=0;i<lin;i++){
        for(j=0;j<col;j++){
            printf("%d\n",*(vet + col*i + j));
        }
    }
    return EXIT_SUCCESS;
```

Vazamento de memória

Exemplo: memory leak

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int *mat,*vet;
7      int lin=512,col=512,i,j;
8      int tecla=1;
9
10
11
12     while(tecla!=0){
13         mat=(int *)calloc(lin*col,sizeof(int));
14
15         printf("Digite: %d\n",tecla);
16         scanf("%d",&tecla);
17         //free(mat);
18
19     }
20
21     return EXIT_SUCCESS;
```