Alocação Dinâmica e Ponteiros

Professores(as):
Virgínia Fernandes Mota
João Eduardo Montandon de Araujo Filho
Leandro Maia Silva

Alocação Dinâmica

- 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 das informações, podendo ser determinada em tempo de execução conforme a necessidade do programa.
- Dessa forma **evita-se o desperdício de memória**.

Alocação Dinâmica

- No padrão C ANSI existem 4 funções para alocações dinâmica pertencentes a biblioteca *stdlib.h*.
- São elas: **malloc()**, **calloc()**, **realloc()** e **free()**.
- Existem ainda outras que não serão abordadas nesta aula, pois não são funções muito utilizadas.

Utilização e funções

- A alocação dinâmica é muito utilizada em problemas de estrutura de dados, por exemplo, listas encadeadas, pilhas, filas, arvores binárias e grafos (AEDS).
- **malloc()** e **calloc()**: são responsáveis por alocar memória;
- **realloc()**: responsável por realocar a memória;
- **free()**: responsável por liberar a memória alocada.

A função **malloc()**

Protótipo:

```
1 void * malloc(unsigned int size);
```

Forma de uso mais comum:

```
1 tipo *variavel;
2
3 variavel = (tipo *) malloc (tamanho * sizeof(tipo));
```

- Esta função recebe como parâmetro "size" que é o número de bytes de memória que se deseja alocar.
- O interessante é que esta função retorna um **ponteiro** do tipo *void* podendo assim ser atribuído a qualquer tipo de *ponteiro*.


POINTERS
memegen.com

Ponteiros

- O ponteiro nada mais é do que uma variável que guarda um endereço de memória (pode ser o endereço de uma variável, mas não necessariamente).
- Declaração:
  tipo * nome; // Esse * indica que é um ponteiro.
- Como atribuir valor ao ponteiro declarado?
  nome = endereçoDeMemoria;
- Como atribuir valor ao local apontado pelo ponteiro?
  ***nome = valor;**
- Usamos o operador & para obter o endereço de memória onde está localizado a variável desejada.
- Usamos o operador * para retornar/atribuir o valor da variável (conteúdo) que está endereçada pelo ponteiro.
- Podem existir ainda ponteiros de ponteiros!
  tipo **nome;

Motivação

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

#define SUCESSO 0

int main(int argc, char ** argv) {
    int num_elementos, vet[100] , i;
    printf("Digite a quantidade de elementos que deseja: ");
    scanf("%d", &num_elementos);
    for ( i = 0; i < num_elementos; i++){
        printf("\n Digite o elemento da posicao %d: ", i);
        scanf("%d", &vet[i]);
    }
    return SUCESSO;
}
```

- E se o usuário colocar apenas 3 elementos no vetor? Desperdício!
- E se o usuário colocar mais de 100 elementos no vetor? Estouro de memória!

Solução

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

#define SUCESSO 0

int main(int argc, char ** argv) {
    int num_elementos, i;
    int *vet; //utilizando um ponteiro

    printf("Digite a quantidade de elementos que deseja: ");
    scanf("%d", &num_elementos);

    //Alocando apenas o espaco necessario
    vet = (int *) malloc (num_elementos * sizeof(int));

    for ( i = 0; i < num_elementos; i++){
        printf("\n Digite o elemento da posicao %d: ", i);
        scanf("%d", &vet[i]);
    }
    return SUCESSO;
}
```

- Agora o usuário pode digitar o tamanho do vetor que desejar e não haverá desperdício de memória!
- Mas ainda temos um problema: Sempre que alocamos dinamicamente a memória precisamos obrigatoriamente liberar essa memória.
    - Na alocação estática isso é feito automaticamente.

A função **free()**

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

#define SUCESSO 0

int main(int argc, char ** argv) {
    int num_elementos, i;
    int *vet; //utilizando um ponteiro

    printf("Digite a quantidade de elementos que deseja: ");
    scanf("%d", &num\_elementos);

    //Alocando apenas o espaco necessario
    vet = (int *) malloc (num_elementos * sizeof(int));

    for ( i = 0; i <  num_elementos; i++){
        printf("\n Digite o elemento da posicao %d: ", i);
        scanf("%d", &vet[i]);
    }

    free(vet);  // libera espaco reservado dinamicamente
    return SUCESSO;
}
```

Atenção!!

`free()` deve ser chamado quando o ponteiro não for mais utilizado.

A função **calloc()**

Protótipo:

```
1 void * calloc(unsigned int num, unsigned int size);
```

Forma de uso mais comum:

```
1 tipo *variavel;
2 variavel = (tipo *) calloc (tamanho, sizeof(tipo));
```

- Esta função inicia o espaço alocado com 0.
- No exemplo anterior, a alocação ficaria:
  vet = (int *) calloc (num_elementos, sizeof(int));

A função **realloc()**

Protótipo:

```
1 void * realloc(void * prt, unsigned int size);
```

Forma de uso mais comum:

```
1 tipo *variavel;
2 variavel = (tipo *) realloc (variavel, tamanho);
```

- Esta função altera o tamanho da memória anteriormente alocado.
- É comum que o endereço onde estava os dados anteriormente seja alterado, por isso o novo endereço é retornado.
- No exemplo anterior, a realocação para um novo_tamanho ficaria:
  vet = (int *) realloc (vet, novo_tamanho);

Alocação dinâmica e Subrotinas

```c
#include <stdio.h>
#include <stdlib.h>
float* alocaVetor(int n){
    return (float *) malloc (n * sizeof(float));
}
void leVetor(float *v, int n){
    int i;
    for (i = 0; i < n; i++)
        scanf("%f", &v[i]);
}

#define SUCESSO 0

int main(int argc, char ** argv) {
    int n;
    float *v;
    printf("\n Digite o numero de elementos do vetor \n");
    scanf("%d", &n);
    v = alocaVetor(n);
    printf("\n Digite seu vetor \n");
    leVetor(v, n);
    free(v);
    return SUCESSO;
}
```

Não se pode retornar vetores em função, mas pode-se retornar ponteiros!

Alocação dinâmica de matrizes

- A alocação dinâmica de memória para matrizes é realizada da mesma forma que para vetores, com a diferença que teremos um ponteiro apontando para outro ponteiro que aponta para o valor final (indireção múltipla).
- Um exemplo de implementação para matriz real bidimensional m x n é fornecido a seguir. A estrutura de dados utilizada neste exemplo é composta por um vetor de ponteiros (correspondendo ao primeiro índice da matriz), sendo que cada ponteiro aponta para o início de uma linha da matriz. Em cada linha existe um vetor alocado dinamicamente (compondo o segundo índice da matriz).

Alocação dinâmica de matrizes I

```c
/* Inclusões */
#include <stdio.h>
#include <stdlib.h>

/* Constantes */
#define SUCESSO                 0
#define MEMORIA_INSUFICIENTE    1
#define PARAMENTRO_INVALIDO     2

int main(int argc, char ** argv) {
    int m, n, i;
    float **matriz;

    printf("\n Digite o numero de linhas e colunas: ");
    scanf("%d %d", &m, &n);

    // verifica parametros recebidos
    if (m < 1 || n < 1) {
        printf ("** Erro: Parametro invalido **\n");
        return PARAMENTRO_INVALIDO;
    }

    // aloca as linhas da matriz
    matriz = (float **) malloc (m*sizeof(float *));
    if (matriz == NULL){
        printf ("** Erro: Memoria Insuficiente **");
        return MEMORIA_INSUFICIENTE;
    }


```

Alocação dinâmica de matrizes II

```
31      // aloca as colunas da matriz
32      for ( i = 0; i < m; i++ ){
33          matriz[i] = (float*) malloc (n * sizeof(float));
34          if (matriz[i] == NULL){
35              printf ("** Erro: Memoria Insuficiente **");
36              return MEMORIA_INSUFICIENTE;
37          }
38      }
39
40      // libera as linhas da matriz
41      for (i = 0; i < m; i++) free (matriz[i]);
42
43      // libera a matriz
44      free(matriz);
45
46      // Se chegou até aqui é porque correu tudo bem
47      return SUCESSO;
48  }
```

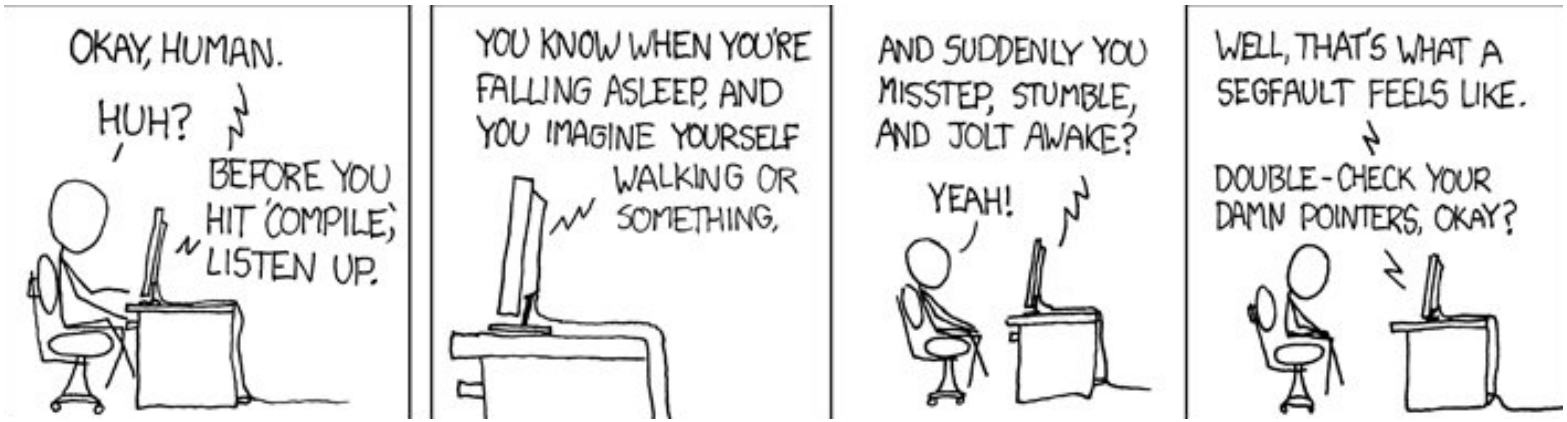
Alocação dinâmica de matrizes

- Este método aloca uma matriz bidimensional dinamicamente. Para se aumentar o número de dimensões basta aumentar o número de ponteiros, ou seja, para 3 dimensões teríamos a seguinte eclaração:
  *****matriz;**
- ... e assim por diante.

Resumindo

- A alocação dinâmica reduz o desperdício de memória e torna o programa mais portável.
- As funções mais utilizadas: **malloc()**, **calloc()**, **realloc()** e **free()**.
- Ponteiros: variáveis que armazenam endereço de memória
- Matrizes podem ser representadas como ponteiros de ponteiros
- Ponteiros podem ser retornados de funções

Resumindo


OKAY, HUMAN.
HUH?
BEFORE YOU HIT `COMPILE', LISTEN UP.
YOU KNOW WHEN YOU'RE FALLING ASLEEP, AND YOU IMAGINE YOURSELF WALKING OR SOMETHING.
AND SUDDENLY YOU MISSTEP, STUMBLE, AND JOLT AWAKE?
YEAH!
WELL, THAT'S WHAT A SEGFAULT FEELS LIKE.
DOUBLE-CHECK YOUR DAMN POINTERS, OKAY?

Exercícios I

1. Implemente uma função que aloque dinamicamente um vetor de inteiros. Essa função deverá receber como parâmetro um inteiro representando o tamanho do vetor, e retornar um ponteiro representando o vetor alocado.
2. Implemente uma função que aloque dinamicamente uma matriz de números reais. Essa função deverá receber como parâmetro dois inteiros representando o tamanho da matriz, e retornar um ponteiro de ponteiros representando a matriz alocada.
3. Facÿa um programa que leia um vetor de um tamanho escolhido pelo usuário e calcule a média aritmética de seus valores.
4. Faça um programa que leia um vetor de números reais de tamanho escolhido pelo usuário e descubra qual é o aior e menor valor existente no vetor, junto de seu índice.

Exercícios II

5. Implemente um programa que receba uma matriz de números reais e retorne a soma dos elementos desta matriz. A matriz deverá ser alocada dinamicamente.
6. Faça um programa para ler a quantidade de um total de X produtos que uma empresa tem em suas Y lojas e imprimir em uma tabela:
    1. o total de cada produto nestas lojas
    2. a loja que tem menos produtos