ALOCAÇÃO DINÂMICA

Estrutura de Dados

Alocação Estática

- Na alocação estática de memória, os tipos de dados tem tamanho predefinido. Neste caso, o compilador vai alocar de forma automática o espaço de memória necessário.
- Sendo assim, dizemos que a alocação estática é feita em tempo de compilação. Este tipo de alocação tende a desperdiçar recursos, já que nem sempre é possível determinar previamente qual é o espaço necessário para armazenar as informações.
- Quando não se conhece o espaço total necessário, a tendência é o programador exagerar pois é melhor superdimensionar do que faltar espaço.

Alocação Dinâmica

- Na alocação dinâmica podemos alocar espaços durante a execução de um programa, ou seja, a alocação dinâmica é feita em tempo de execução. Isto é bem interessante do ponto de vista do programador, pois permite que o espaço em memória seja alocado apenas quando necessário.
- Além disso, a alocação dinâmica permite aumentar ou até diminuir a quantidade de memória alocada.

Alocação Dinâmica

- **sizeof**
  - A função sizeof determina o número de bytes para um determinado tipo de dados.
  - É interessante notar que o número de bytes reservados pode variar de acordo com o compilador utilizado.
  - Exemplo:
    - x = sizeof(int); //retorna 4 no gcc
- **malloc**
  - A função malloc aloca um espaço de memória e retorna um ponteiro do tipo void para o início do espaço de memória alocado.
- **free**
  - A função free libera o espaço de memória alocado.

Alocação Dinâmica

- Quando um programador define tipo e o número de elementos um vetor ele está utilizando alocação estática.
- Uma alternativa interessante é declarar um vetor como ponteiro, a fim de utilizar alocação dinâmica. Para tanto devemos usar a função malloc. Porém, esta função necessita saber a quantidade de bytes que devem ser reservados. Para fazer esse cálculo usamos o comando sizeof.

Alocação Dinâmica

```
#include <stdio.h>
//necessário para usar as funções malloc() e free()
#include <stdlib.h>
#include <conio.h>
int main(void)
{
//definindo o ponteiro v
     float *v;
     int i, num_componentes;
     printf("Informe o numero de componentes do vetor\n");
     scanf("%d", &num_componentes);
```

Alocação Dinâmica

1. Calcular o número de bytes necessários primeiramente multiplicamos o número de componentes do vetor pela quantidade de bytes que é dada pelo comando sizeof, portanto temos:
   - num_componentes * sizeof(float)

2. Reservar a quantidade de memória usamos malloc para reservar essa quantidade de memória, então temos:
   - malloc(num_componentes * sizeof(float))

3. Converter o ponteiro para o tipo de dados desejado como a função malloc retorna um ponteiro do tipo void, precisamos converter esse ponteiro para o tipo da nossa variável, no caso float, por isso usamos o comando de conversão explicita:
   - (float *)

Alocação Dinâmica

```
v = (float *) malloc(num_componentes * sizeof(float));
//Armazenando os dados em um vetor
  for (i = 0; i < num_componentes; i++)
  {
    printf("\nDigite o valor %d do vetor: ", i+1);
    scanf("%f",&v[i]);
  }
  printf("\n**** Valores do vetor dinamico ****\n\n");
  for (i = 0;i < num_componentes; i++)
  {
    printf("%.2f\n",v[i]);
  }
  free(v);
```

Redimensionamento realloc

- Às vezes é necessário alterar, durante a execução do programa, o tamanho de um bloco de bytes que foi alocado por malloc.
- A função realloc recebe o endereço de um bloco previamente alocado por malloc (ou por realloc) e o número de bytes que o bloco redimensionado deve ter.
- A função aloca o novo bloco, copia para ele o conteúdo do bloco original, e devolve o endereço do novo bloco.

Redimensionamento realloc

- Se o novo bloco for uma extensão do bloco original, seu endereço é o mesmo do original (e o conteúdo do original não precisa ser copiado para o novo).
  - Caso contrário, realloc copia o conteúdo do bloco original para o novo e libera o bloco original (invocando free). A propósito, o tamanho do novo bloco pode ser menor que o do bloco original.
- Suponha, por exemplo, que alocamos um vetor de 1000 inteiros e depois decidimos que precisamos de duas vezes mais espaço. Veja um caso concreto:

Redimensionamento realloc

```
int *v;
  v = malloc (1000 * sizeof (int));
  for (int i = 0; i < 990; i++)
     scanf ("%d", &v[i]);
  v = realloc (v, 2000 * sizeof (int));
  for (int i = 990; i < 2000; i++)
     scanf ("%d", &v[i]);
```

A memória é finita

Se a memória do computador já estiver toda ocupada, malloc não consegue alocar mais espaço e devolve NULL. Convém verificar essa possibilidade antes de prosseguir:

```
ptr = malloc (sizeof (data));
if (ptr == NULL) {
   printf ("Socorro! malloc devolveu NULL!\n");
   exit (EXIT_FAILURE);
}
```