MALLOC, CALLOC e REALLOC

Muito bem. Agora que você entendeu o funcionamento da alocação dinâmica de memória e como podemos usar a função `malloc()`, é fácil entender o funcionamento da `calloc()` e da `realloc()`. Vejamos:

MALLOC

Como vimos, a função `malloc()` serve para alocar memória e tem o seguinte protótipo com um argumento:

```
void *malloc (unsigned int num);
```

Malloc toma o número de bytes que desejamos alocar (num), aloca na memória e retorna um ponteiro void * para o primeiro byte alocado. O ponteiro void * pode ser atribuído a qualquer tipo de ponteiro. Se não houver memória suficiente para alocar a memória requisitada a função `malloc()` retorna um ponteiro nulo. Analise o código da figura 1, onde a memória necessária para `vet1` é alocada com `malloc()`. Reveja o vídeo da aula caso ainda tenha alguma dúvida.

CALLOC

A função `calloc()` também serve para alocar memória, mas seu protótipo é um pouco diferente (dois argumentos):

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

Como você pode observar, a função calloc recebe dois parâmetros: a quantidade de elementos que iremos salvar na região de memória alocada e o tamanho em memória de um elemento desse tipo.

Um detalhe importante é que **calloc garante que o espaço alocado é inicializado com zeros**, enquanto que malloc não faz isso - ou seja, a área reservada com malloc pode conter “sujeira" de memória.

Se não houver memória suficiente para alocar a memória requisitada a função `calloc()` retorna um ponteiro nulo. Veja um exemplo no código da figura 1, onde a memória necessária para `vet2` é alocada com `calloc()`.

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

int main(void){
    int tam=10;
    int *vet1, *vet2;

    // alocar memória com MALLOC não limpa o conteúdo anterior da memória
    // processamento mais rápido
    vet1 = (int*)malloc(tam * sizeof(int));

    printf("Vetor criado com MALLOC:\n");
    for (int i = 0; i < tam; i++) {
        printf("Elemento %d = %4d \n", i, vet1[i]);
    }

    // alocar memória com CALLOC zera o conteúdo anterior da memória
    // processamento mais lento
    vet2 = (int*)calloc(tam, sizeof(int));
    printf("\nVetor criado com CALLOC:\n");
    for (int i = 0; i < tam; i++) {
        printf("Elemento %d = %4d \n", i, vet2[i]);
    }

    free(vet1);
    free(vet2);
}
```

```
Vetor criado com MALLOC:
Elemento 0 =    0
Elemento 1 = -1342177280
Elemento 2 =    0
Elemento 3 = -1342177280
Elemento 4 = -1213988848
Elemento 5 = 32767
Elemento 6 = 2124307720
Elemento 7 = 32767
Elemento 8 = -1213931160
Elemento 9 = 32767

Vetor criado com CALLOC:
Elemento 0 =    0
Elemento 1 =    0
Elemento 2 =    0
Elemento 3 =    0
Elemento 4 =    0
Elemento 5 =    0
Elemento 6 =    0
Elemento 7 =    0
Elemento 8 =    0
Elemento 9 =    0
```

Figura 1: *alocação de memória com malloc e calloc. Note, no resultado da execução (direita), que a memória alocada com malloc pode possuir “sujeira” de memória, diferente da alocada com calloc que está inicializada com zeros.* ***Atenção:*** *este código não está verificando se a alocação foi realizada com sucesso. Foi feito assim propositalmente para reduzir o tamanho do exemplo. Não esqueça de incluir essa parte no seu código, como visto em aula.*

REALLOC

A função `realloc()` serve para **realocar** memória e tem o seguinte protótipo (dois argumentos):

```
void *realloc (void *ptr, unsigned int num);
```

A função **modifica o tamanho da memória previamente alocada** apontada por `*ptr` para aquele especificado por `num`. O valor de num pode ser maior ou menor que o original. Um ponteiro para o bloco de memória é devolvido porque `realloc()` pode precisar mover o bloco para aumentar seu tamanho. Se isso ocorrer, o conteúdo do bloco antigo é copiado no novo bloco e nenhuma informação é perdida.

Se `ptr` for nulo, `realloc()` aloca `size` bytes e devolve um ponteiro; se `size` é zero, a memória apontada por ptr é liberada. Se não houver memória suficiente para a alocação, um ponteiro nulo é devolvido e o bloco original é deixado inalterado.

Analise o código da figura 2, no qual é realizada alocação de memória para a variável `vet`, inicialmente com 10 espaços (`tam = 10`). Posteriormente, são realocados mais 10 espaços de memória (`tam = 20`). Note que após a realocação não são perdidos os dados previamente armazenados na variável - o segundo laço do código apenas complementa os valores armazenados de onze em diante.

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

int main(void){
    int tam = 10;
    int *vet, len;

    vet = (int*)malloc(tam * sizeof(int));

    //criaçao de vetor (pode ser com MALLOC ou CALLOC)
    printf("Vetor criado com MALLOC:\n");
    for (int i = 0; i < tam; i++) {
        vet[i]=i+1;
        printf("Elemento %d = %4d \n", i, vet[i]);
    }

    tam = 20;

    // uso de REALLOC para modificar o tamanho do vetor
    vet = (int*)realloc(vet, tam * sizeof(int));

    printf("\nVetor modificado com REALLOC:\n");
    for (int i = 0; i < tam; i++) {
        if (i > 9){
            vet[i]=i+1;
        }
        printf("Elemento %d = %4d \n", i, vet[i]);
    }

    free(vet);
}
```

```
Vetor criado com MALLOC:
Elemento 0 =    1
Elemento 1 =    2
Elemento 2 =    3
Elemento 3 =    4
Elemento 4 =    5
Elemento 5 =    6
Elemento 6 =    7
Elemento 7 =    8
Elemento 8 =    9
Elemento 9 =   10

Vetor modificado com REALLOC:
Elemento 0 =    1
Elemento 1 =    2
Elemento 2 =    3
Elemento 3 =    4
Elemento 4 =    5
Elemento 5 =    6
Elemento 6 =    7
Elemento 7 =    8
Elemento 8 =    9
Elemento 9 =   10
Elemento 10 =   11
Elemento 11 =   12
Elemento 12 =   13
Elemento 13 =   14
Elemento 14 =   15
Elemento 15 =   16
Elemento 16 =   17
Elemento 17 =   18
Elemento 18 =   19
Elemento 19 =   20
```

Figura 2: *alocação de memória com malloc e alteração do tamanho de memória alocada em tempo de execução com realloc. Note no resultado da execução (direita) que, após o realloc, o segundo laço apenas complementa o preenchimento dos dados.* ***Atenção:*** *este código não está verificando se a alocação/ realocação foi realizada com sucesso. Foi feito assim propositalmente para reduzir o tamanho do exemplo. Não esqueça de incluir essa parte no seu código, como visto em aula.*

Transcreva os códigos de exemplo e realize alterações neles para verificar sua compreensão sobre o assunto. Após isso, resolva a lista de exercícios disponibilizada.

LISTA DE EXERCÍCIOS

Alocação Dinâmica de Memória

Atividade avaliativa

1 Construa um programa que receba do usuário o tamanho de um vetor de inteiros a ser lido e efetue a alocação dinâmica de memória. A seguir, leia as entradas do usuário para preencher o vetor e exiba o vetor resultante.

2 Elabore um programa que simule a memória de um computador. O usuário irá especificar o tamanho da memória, ou seja, quantos bytes serão alocados do tipo inteiro (a memória deve ser inicializada com todos os dados zerados). Assim, a memória solicitada deve ser um valor múltiplo do tamanho do tipo inteiro. A seguir, o usuário terá duas possibilidades mediante um menu: inserir um valor em uma determinada posição ou consultar o valor contido em uma determinada posição.

3 Desenvolva um programa que armazene na memória dois vetores do tipo inteiro contendo 1000 posições cada um. O primeiro deve ser criado utilizando MALLOC, e o segundo utilizando CALLOC. Após isso, percorra os vetores com uma função que retorne a quantidade de zeros existentes em cada um deles.

4 Faça um programa que leia uma quantidade qualquer de números armazenando-os na memória e pare a leitura quando o usuário entrar um valor negativo. Em seguida, imprima o vetor lido. Use a função REALLOC como achar mais conveniente para realocar memória sempre que necessário.

5 Crie um programa que declare uma estrutura (registro/struct) para um cadastro de alunos segundo os seguintes requisitos:

a) Deverão ser armazenados para cada aluno: matrícula, nome (apenas um) e ano de nascimento.
b) No início do programa o usuário deverá informar o total de alunos que serão armazenados e o programa deverá alocar dinamicamente a quantidade necessária de memória para armazenar os registros dos alunos.
c) O programa deve solicitar ao usuário a entrada de dados dos alunos.
d) Ao final do cadastramento, o programa deve exibir os dados armazenados e liberar a memória.

6 Escreva um programa que aloque dinamicamente uma matriz de inteiros de dimensões definidas pelo usuário e a preencha de forma automática com valores aleatórios entre 0 e 100. Em seguida, implemente uma função que receba um valor digitado pelo usuário e retorne 1 (um) caso o valor informado esteja na matriz ou retorne 0 (zero) caso o valor não seja localizado.

7 Crie um programa que receba do usuário dois números N e M e crie uma matriz de inteiros N x M. A seguir a matriz deve ser preenchida com valores aleatórios de forma automática. Por fim, o programa deve localizar os três maiores números contidos na matriz e exibir os índices da linha e coluna onde se encontra cada um deles.

8 Construa um programa que leia números do teclado e os armazene em um vetor alocado dinamicamente. O usuário irá digitar uma sequência de números sem limite de quantidade. Os números serão digitados um a um até que o usuário entre o número 0 (zero) para encerrar a entrada de dados. Para o armazenamento, o programa deve atender os seguintes requisitos:

a) Inicia com um vetor de tamanho 10 alocado dinamicamente (n = 10).
b) Ao encher o vetor com as entradas do usuário, deve ser alocado um novo vetor com o tamanho do vetor anterior mais 10 posições (n += 10).
c) Copie os valores já digitados da área de memória inicial para esta área maior e libere a memória da área inicial.
d) Repita esse procedimento de expandir dinamicamente com mais 10 valores o vetor alocado cada vez que o mesmo estiver cheio - dessa forma o vetor irá se expandindo de 10 em 10 valores.

Ao final, exiba o vetor lido. ATENÇÃO: não use REALLOC.

Bom estudo!