

Alocação de Memória

Todo programa precisa utilizar memória para ser executado. Quando um programa inicia sua execução, ele começa a solicitar memória ao sistema operacional, ou seja, faz a alocação de memória necessária para a sua execução. Nem sempre a memória alocada na iniciação do programa é suficiente, então o programa também precisa alocar memória durante a sua execução.

A linguagem C permite dois tipos de alocação de memória: Alocação estática e alocação dinâmica.

Alocação Estática de Memória

Na alocação estática o espaço de memória, que as variáveis irão utilizar durante a execução do programa, **é definido no processo de compilação**. Não sendo possível alterar o tamanho desse espaço durante a execução do programa.

Exemplos:

```
/*Espaço reservado para um valor do tipo char. O
char ocupa 1 byte na memória.*/
char a;
```

```
/*Espaço reservado para dez valores do tipo int.
O int ocupa 4 bytes na memória, portanto 4x10=40 bytes.*/
int vetor[10];
```

```
/*Espaço reservado para nove(3x3) valores do tipo
double. O double ocupa 8 bytes na memória, portanto
3x3x8=72 bytes.*/
double matriz[3][3];
```

Este tipo de alocação é utilizado quando se sabe de antemão a quantidade de memória que será utilizada pelo programa.

Alocação Dinâmica de Memória

Na alocação dinâmica o espaço de memória, que as variáveis irão utilizar durante a execução do programa, **é definido enquanto o programa está em execução**. Ou seja, quando não se sabe ao certo quanto de memória será necessário para o armazenamento das informações, podendo ser determinadas, sob demanda, em tempo de execução conforme a necessidade do programa.

No padrão C ANSI existem quatro funções para alocação dinâmica de memória:

1. malloc()
2. calloc()
3. realloc()
4. free()

Todas elas pertencem a biblioteca <stdlib.h>. Iremos nos concentrar nas funções malloc() e free(), pois são as mais utilizadas.

Sintaxe da função malloc():

```
void *malloc(size_t num_bytes);
```

Esta função recebe como parâmetro "num_bytes" que é o número de bytes de memória que se deseja alocar. O tipo size_t é definido em stdlib.h como sendo um inteiro sem sinal. O interessante é que esta função retorna um **ponteiro** do tipo void podendo assim ser atribuído a qualquer tipo de ponteiro.

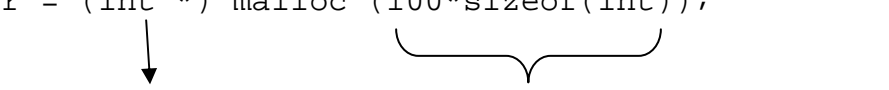
Exmeplo:

```
int *vetor;  
vetor = malloc(100);
```

No exemplo acima foram alocados, dinamicamente, cem bytes da memória Heap. Quando os programas alocam memória dinamicamente, a biblioteca de execução C recebe memória a partir de um reservatório de memória livre (não utilizado) chamado Heap.

A função malloc() devolve um ponteiro do tipo void, desta forma pode-se atribuí-lo a qualquer tipo de ponteiro. Portanto, precisamos fazer uma conversão (cast) para o tipo desejado e também alocar um espaço compatível com o tipo de destino. A alocação para atender a essas necessidades deve ser feita desta forma:

```
vetor = (int *) malloc (100*sizeof(int));
```



Conversão para o tipo
int *

Serão alocados 400 bytes no total: 100 x 4.

Como a memória ocupada por um determinado tipo pode variar de máquina para máquina, devemos utilizar o operador sizeof, que retorna o tamanho do parâmetro informado.

Como foi descrito anteriormente, a memória alocada é obtida da memória Heap. Contudo, não há garantias de que a Heap possua memória disponível para o seu programa. Portanto, é necessário verificar se existe espaço suficiente na Heap para alocar a memória desejada. Para atender a este requisito, devemos fazer um teste para verificar se a memória foi realmente alocada.

Exemplo:

```
vetor = (int *) malloc (100*sizeof(int));
if (vetor == NULL){
    printf ("Não há memória suficiente para alocação");
    exit(1);
}
```

Quando a função malloc não consegue alocar memória, pois não há espaço suficiente na Heap, a mesma retorna NULL, indicando que a requisição de memória foi negada. Quando a alocação é feita com sucesso, malloc retorna um ponteiro para a região de memória alocada.

Sintaxe da função free():

```
void free (void *ptr);
```

Quando o programa não precisa mais da memória que foi alocada pela função malloc, ele deve liberar esta memória, ou seja, informar ao Sistema Operacional que aquela região de memória não será mais utilizada. Para liberar memória alocada, devemos utilizar a função free.

Na sintaxe da função free, o parâmetro ptr é um ponteiro para o início da região de memória que se quer liberar.

Além da função malloc, a linguagem C possui outra função para alocação dinâmica de memória: calloc.

Sintaxe da função calloc():

```
void *calloc(size_t num_itens, size_t tam_item);
```

O parâmetro num_itens especifica quantos elementos calloc precisa alocar. Já o parâmetro tam_item especifica o tamanho, em bytes, de cada um desses elementos.

Por fim temos a função `realloc`, que é responsável por alterar o tamanho de um bloco de memória previamente alocado.

Sintaxe da função `realloc()`:

```
void *realloc(void *ptr, size_t tam);
```

O parâmetro `ptr` representa a região de memória que se deseja alterar. Já o parâmetro `tam` representa o novo tamanho da memória apontada por `ptr`. Este valor de `tam` pode ser maior ou menor que o original.

Diferenças, Vantagens e Desvantagens Entre Alocação Estática e Alocação Dinâmica.

Na alocação estática, o espaço de memória é definido durante o processo de compilação, já na alocação dinâmica o espaço de memória é reservado durante a execução do programa.

Na alocação estática não é possível alterar o tamanho do espaço de memória que foi definido durante a compilação, já na alocação dinâmica este espaço pode ser alterado dinamicamente durante a execução.

A alocação estática tem a vantagem de manter os dados organizados na memória, dispostos um ao lado do outro de forma linear e sequencial. Isto facilita a sua localização e manipulação, em contrapartida, precisamos estabelecer previamente a quantidade máxima necessária de memória para armazenar uma determinada estrutura de dados.

Exemplo:

```
int vetor[5] = {10, 50, 35, 45, 60};
```

22FF20	22FF24	22FF28	22FF2C	22FF30
10	50	35	45	60

Como já foi descrito, a quantidade de memória que se deve alocar estaticamente é definida durante a compilação, portanto corre-se o risco de sub ou superestimar a quantidade de memória alocada. Isso não acontece na alocação dinâmica, pois como a alocação é feita durante a execução, sabe-se exatamente a quantidade necessária. Isso permite otimizar o uso da memória.

A alocação dinâmica é feita por meio de funções de alocação e liberação de memória e é de responsabilidade do programador usar essas funções de forma coerente, pois o seu uso incorreto pode causar efeitos colaterais indesejados no programa como GPF e vazamento de memória.

Na alocação dinâmica podemos redimensionar uma área previamente alocada, já na alocação estática isso não é possível, pois o tamanho foi definido durante a compilação.