VOLTAR

Alocação dinâmica de memória

Apresentar os conceitos principais de alocação dinâmica em C e alguns exemplos da aplicação desse recurso.

NESTE TÓPICO

Introdução

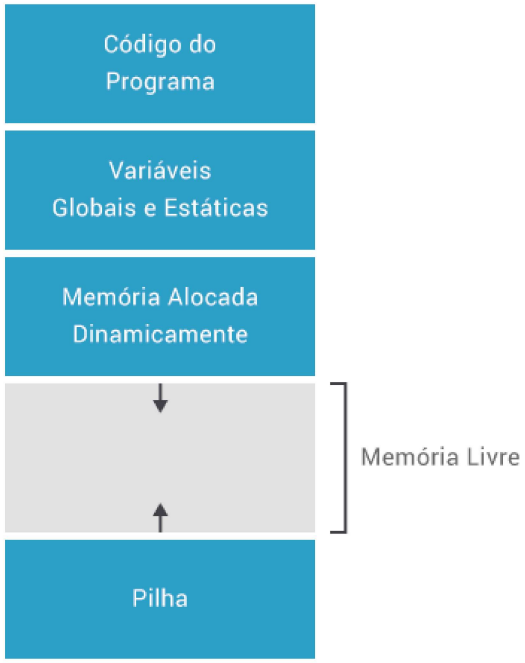
Basicamente, existem três maneiras de reservar espaço de memória para o armazenamento de dados: o uso de variáveis globais, o uso de variáveis locais e a alocação dinâmica de variáveis.

Para as variáveis globais é reservado espaço de memória que existe enquanto o programa estiver executando. No caso das variáveis locais, o espaço existe apenas enquanto a função que declarou a variável está sendo executada, sendo liberado para outros usos quando a execução da função termina. Por esse motivo, a função que chama não pode fazer referência ao espaço local da função chamada.

A alocação dinâmica é o processo que aloca memória em tempo de execução. Esse espaço alocado dinamicamente permanece reservado até que seja explicitamente liberado pelo programa. Ela é utilizada quando não se sabe ao certo quanto de memória será necessário para o armazenamento das informações, podendo ser determinadas em tempo de execução, conforme a necessidade do programa. Dessa forma, evita-se o desperdício de memória. A alocação dinâmica é muito utilizada em problemas de estrutura de dados como, por exemplo, listas encadeadas, pilhas, filas, arvores binárias e grafos.

Na Figura abaixo é possível observar, de maneira fictícia, a distribuição do uso da memória pelo sistema operacional. Quando um programa é executado, o sistema operacional reserva espaços de memória necessários para armazenar as variáveis globais (estáticas). O restante da memória livre

é utilizado pelas variáveis locais e pelas variáveis alocadas dinamicamente. Cada vez que uma determinada função é chamada, o sistema reserva o espaço necessário para as variáveis locais da função. Esse espaço pertence à pilha de execução e, quando a função termina, é desempilhado.



Simulação da alocação esquemática de memória pelo sistema operacional.

Fonte: CELES, W. e outros. Introdução a estruturas de dados: com técnicas de programação em C. Rio de Janeiro, Elsevier, 2004.

Na linguagem C, para trabalhar com a alocação dinâmica de memória, é necessário utilizar funções da biblioteca "stdlib.h", a qual será abordada a seguir.

Bibiblioteca "stdlib.h"

A biblioteca padrão “stdlib.h” contém uma série de funções pré-definidas para tratar alocação dinâmica de memória em C. Abaixo, seguem as funções mais utilizadas:

- malloc()
- sizeof()
- realloc()
- free()

1. Função *malloc*:

A função *malloc* reserva espaço de memória livre. Esta função recebe como parâmetro o número de bytes de memória que se deseja alocar. A função retorna um ponteiro (endereço) genérico (*void**) para o endereço inicial da área da memória alocada se houver espaço livre. Caso contrário, retorna um endereço nulo (representado pelo símbolo NULL), se não houver espaço livre.

A sintaxe da função é a seguinte:

```
1.  void * malloc(number_bytes); //número de bytes alocados do formato inteiro sem sinal
```

No exemplo abaixo, suponha que seja necessário, no meio do código, alocar um espaço de memória com 150 bytes. Para que isso aconteça, é necessário digitar as seguintes linhas de código:

```
1.  char  *str;
2.  str = malloc(150);
```

Para exemplificar, vamos considerar a alocação dinâmica de um vetor de inteiro com 20 elementos. Como a função *malloc* tem como valor de retorno o endereço da área alocada e, nesse exemplo, desejamos armazenar valores inteiros nessa área, devemos declarar um ponteiro de inteiro para receber o endereço inicial do espaço alocado. Abaixo, segue a codificação.

```
1.  int *v;
2.
3.  v = malloc (20*4);
```

Após esse comando, se a alocação for bem-sucedida, v armazenará o endereço inicial de uma área contínua de memória suficiente para armazenar 20 valores inteiros. Nesse caso, podemos tratar o vetor v como se ele estivesse declarado estaticamente, pois se v aponta para o início da área alocada, então v [0] acessa o espaço para o primeiro elemento armazenado, v [1] acessa o segundo e assim sucessivamente.

É importante salientar que a função *malloc* é usada para alocar espaço para armazenar valores de qualquer tipo. Por isso, *malloc* retorna um ponteiro genérico, para um tipo qualquer, representado por *void**. No entanto, é comum fazer a conversão explicitamente, utilizando o operador de molde de tipo (*cast*). A linha de instrução para a alocação do vetor de inteiros fica então:

```
1.  v = (int *) malloc (20*4);
```

O trecho de código abaixo exemplifica o caso em que não há espaço suficiente para realizar a alocação na memória. Assim, podemos imprimir uma mensagem e abortar o programa com a função *exit*, também definida na stdlib.

```
v = (int *) malloc (20*4);

if (v==NULL)
{
   printf ("Memoria insuficiente. \n");
   exit (1); //aborta o programa e retorna 1
}
```

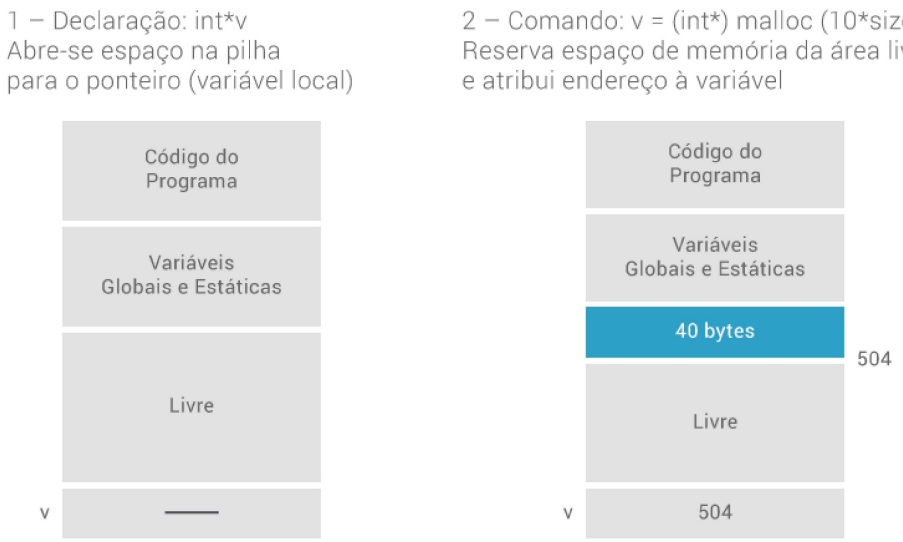
2. Função sizeof:

A função *sizeof* retorna o tamanho, em bytes, do que for definido como parâmetro para a função. A sintaxe da função é a seguinte:

```
sizeof (variavel ou tipo de dados);
```

Na alocação do vetor com 20 números inteiros mostrado anteriormente, consideramos que um inteiro ocupa 4 bytes. Para ficarmos independentes de compiladores e máquinas, usamos o operador *sizeof( )*, como mostrado no exemplo a seguir. A Figura abaixo ilustra, de maneira esquemática, o que ocorre na memória.

```
v = (int *) malloc (20*sizeof(int));
```



Simulação da alocação dinâmica de memória.

Fonte: CELES, W. e outros. Introdução a estruturas de dados: com técnicas de programação em C, Rio de Janeiro, Elsevier, 2004.

3. Função *realloc*:

A função *realloc* é usada para redimensionar um espaço alocado previamente com *malloc*. Seus argumentos são um ponteiro para o inicio de uma área previamente alocada, e o novo tamanho, que pode ser maior ou menor que o tamanho original.

realloc retorna um ponteiro para a nova área alocada. Este ponteiro pode ser igual ao ponteiro original se o novo tamanho for menor que o tamanho original, e diferente do ponteiro original se o novo tamanho for maior que o tamanho original. Neste último caso, *realloc* copia os dados da área original para a nova área. A sintaxe da função é a seguinte:

```
1.  void * realloc (void *p, tamanho_novo);
```

No exemplo abaixo, é mostrada a alocação dinâmica, utilizando a função *malloc*, da variável *frase* como uma string de tamanho 6. Em seguida, com a função *realloc,* é feita a realocação da variável *frase* com tamanho 20.

```
1.  char *frase;
2.  frase=(char *)malloc(6);
3.
4.  strcpy(frase,"teste");
5.
6.  //realocação da variável frase na memoria
7.  frase=(char *)realloc(frase,20);
8.
9.  strcpy(frase,"teste estrutura");
```

4. Função *free*:

As variáveis alocadas dentro de uma função, também conhecidas como variáveis *automáticas* ou *locais*, desaparecem assim que a execução da função termina. Já as variáveis alocadas dinamicamente continuam a existir mesmo depois que a execução da função termina. Se for necessário liberar a memória ocupada por essas variáveis, é preciso recorrer à função *free*.

A função *free* recebe como argumento um ponteiro para uma área de memória previamente alocada por malloc() e então libera esta área para uma possível utilização futura. Ela não deve ser aplicada a uma *parte* de um bloco de bytes alocado por *malloc*. Aplique *free* apenas ao bloco todo. A sintaxe da função é a seguinte:

```
1.  free (variavel);
```

No exemplo abaixo, é mostrada a alocação dinâmica, utilizando a função *malloc*, da variável *teste* e, em seguida, o espaço reservado para a variável é liberado, utilizando a função *free*.

```
1.  float *teste;
2.
3.  teste = (float *) malloc (15*sizeof(float));
4.
5.  free (teste);
```

Exemplo

No programa a seguir, é mostrado um exemplo da alocação de um vetor de número inteiros de tamanho que pode ser definido pelo usuário. Dessa forma, a alocação do espaço de memória para o vetor é feita, dinamicamente, por meio da função *malloc*. Antes de finalizar o programa, a variável é liberada da memória por meio da função *free*.

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.
4.  main()
5.  {
6.      int *vet, tam, i;
7.
8.      printf ("\n Digite o tamanho do vetor:");
9.      scanf("%d",&tam);
10.
11.     vet = (int *) malloc(tam* sizeof (int));
12.
13.     for(i=0;i<tam;i++)
14.     {
15.       printf (" elemento:");
16.       scanf("%d",&vet[i]);
17.     }
18.
19.
20. for(i=0;i<tam;i++)
21.     {
22.       printf ("\n vet[%d]=%d",i,vet[i]);
23.     }
24.     system ("PAUSE");
25.
26.     free (vet);//desalocação do vetor
27. }
```

Quiz

Exercício

Alocação dinâmica de memória

INICIAR

Quiz

Exercício Final

Alocação dinâmica de memória

INICIAR

Referências

MIZRAHI, V. V. *Treinamento em linguagem C*, São Paulo: Pearson, 2008.

SCHILDT, H. C – Completo e Total, São Paulo: Pearson, 2006.

CELES, W. e outros. Introdução a estruturas de dados: com técnicas de programação em C, Rio de Janeiro, Elsevier, 2004.

Avalie este tópico

Ajuda? (https://ava.uninove.br/seu/AVA/topico/topico.php?idCurso=)