Alocação Dinâmica de Memória

Módulo 9 Aula 4

Linguagem C, o Curso Definitivo WR Kits

Autor: Dr. Eng. Wagner Rambo

Imagine que você precisa de um vetor de caracteres, mas não sabe exatamente qual a dimensão. Ela poderá ser de 10 bytes, mas também poderá ser de 1000 bytes. Programadores inexperientes vão declarar este vetor conforme o Box 1.

```
char vector[1000];
```

Box 1 - Vetor com o máximo tamanho previsto para o projeto.

A questão aqui é que o vetor poderá em algum momento utilizar esses 1000 bytes, mas na grande maioria das vezes utilizará menos de 10% disso. No entanto, a memória alocada foi de 1000 bytes e teremos com bastante frequência mais de 900 bytes ociosos. Falando em microcontroladores de memória reduzida, isso é péssimo. Também não é uma boa prática em softwares onde você deseja eficiência.

Para contornar este problema, temos algumas funções na Linguagem C que permitirão alocação dinâmica de memória, que consiste em utilizarmos apenas a memória necessária no tempo de execução. Se em dado momento nosso sistema precisar de 20 bytes, este será o tamanho alocado; se precisar de 900, 50, 2000, enfim, alocaremos apenas a memória necessária. A principal função utilizada para alocação dinâmica é a *malloc* e tem seu protótipo explícito no Box 2.

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

Box 2 - Protótipo da função malloc (memory allocation).

O tipo *size_t* normalmente é definido como *unsigned int* através de *typedef* na biblioteca *stdlib.h*. A função *malloc* basicamente cria um bloco com n bytes retornando assim o endereço desse bloco. Caso a alocação falhe, a função retorna NULL. Observe que o tipo desta função é *void**, repare no operador de indireção presente ali. Por este motivo, a função retorna um ponteiro para o tipo de dado a ser alocado. Portanto *void** significa o retorno de um ponteiro para qualquer tipo de dado.

No Box 3, apresentamos um exemplo clássico de aplicação para alocação dinâmica memória, onde queremos copiar uma *string* em outra. Como a *string* original é preenchida pelo usuário em tempo de execução, podemos alocar a *string* cópia dinamicamente.

```
main()
{
  char s[100],
       *palloc; /* ponteiro para caracteres */

  printf("Entre com a string: ");
  gets(s);

  /* aloca memória */
  palloc = (char *)malloc(strlen(s)+1);

  /* OBS.: incluir a biblioteca string.h no projeto */

  if(palloc==NULL) /* verifique se alocou corretamente */
    puts("Memoria insuficiente\n");

  else
  {
    strcpy(palloc,s);  /* copia strings */

    putchar(0x0A);
    printf(s);
    putchar(0x0A);
    printf(palloc);
    putchar(0x0A);

    free(palloc);      /* libera memória */

  }

} /* end main */
```

Box 3 - Alocando memória dinamicamente para uma string.

Primeiro declaramos uma *string* s com tamanho 99+1 (lembrando que o byte final é reservado para o caractere nulo). Após um ponteiro para caracteres, que será utilizado para alocar a memória dinamicamente. Solicitamos a entrada da *string* e em seguida, com a função *malloc*, alocamos memória de acordo com o tamanho atual de s e mais 1 byte extra, para o caractere nulo. Observe o *casting* para *char** garantindo a compatibilidade com o ponteiro *palloc*. Sempre é conveniente verificarmos se a memória foi alocada corretamente, para isso, basta testarmos se o ponteiro está apontando para NULL. Caso esteja, ocorreu um erro e notificamos isso. Do contrário, alocamos corretamente e iremos copiar a *string* original para o espaço alocado. Observe que utilizamos o ponteiro *palloc* indicando o endereço do primeiro elemento da *string* nova. Por fim, imprimimos ambas as *strings* para verificar se está tudo correto e com a função *free*, liberamos a memória alocada.

Outra grande vantagem da memória dinâmica é quando falamos em softwares que irão ler algum arquivo e realizar um processamento a partir deste. O programa do Box 4 irá ler um arquivo de texto e alocar memória suficiente para exibir o seu conteúdo na tela.

```
main()
{
  FILE *arq;
  unsigned char *palloc;
  unsigned long int num_bytes=0,i=0;
  int chr;

  arq = fopen("code.txt","r");

  if(arq == NULL)
  {
    printf("Falha ao abrir o arquivo.\n");
    system("pause");
    exit(0);
  }

  printf("Arquivo aberto com sucesso!\n");

  while((chr = fgetc(arq))!=EOF)
    num_bytes++;

  printf("Tamanho do arquivo: %lu bytes\n",num_bytes);
  rewind(arq);  /* ponteiro retorna ao início do arquivo */

  /* aloca memória de acordo com tamanho do arquivo */
  palloc = (unsigned char *) malloc(num_bytes*sizeof(char));

  if(palloc == NULL)
  {
    printf("Memoria insuficiente.\n");
    system("pause");
    exit(0);
  }

  /* armazena os caracteres */
  while((chr = fgetc(arq))!=EOF)
  {
    if(i<=num_bytes) palloc[i] = (unsigned char) chr;
    i++;
  }

  for(i=0;i<num_bytes;i++)
    printf("%c",palloc[i]);

  fclose(arq);   /* fecha arquivo */
  free(palloc);  /* libera memória */

} /* end main */
```

Box 4 - Alocando memória dinamicamente para leitura de um arquivo.

O código do Box 4 pode ser bastante útil para entendermos os compiladores, onde poderemos ler todo o código fonte previamente, alocar a memória necessária para gerar o código objeto e criar o arquivo binário resultante caso não existam erros de sintaxe no código fonte.

Outra função de alocação dinâmica é a *calloc*, cujo protótipo é apresentado no Box 5.

```
void *calloc(size_t num, size_t size);
```

Box 5 - Protótipo da função calloc.

A função *calloc* é utilizada para alocar uma quantidade de memória igual ao produto de *num* e *size*. Assim como *malloc*, *calloc* retorna um ponteiro para o endereço do primeiro byte da região de memória alocada. Retornará NULL caso haja falha na alocação. No Box 6 apresentamos uma função que retorna um ponteiro para um vetor de qtd x *longs*.

```
long *vec_long(unsigned qtd)
{
  long *palloc;

  palloc = calloc(qtd, sizeof(long));

  if(!palloc)
  {
    printf("Fatal error\n");
    exit(1);
  }

  return palloc;

}
```

Box 6 - Função para alocar um vetor de longs de forma dinâmica.

O programa do Box 7 poderá ser utilizado para testar a função *vec_long*. No exemplo, alocamos um vetor com 20 dados do tipo *long* e preenchemos com os números de 1 a 20.

```
main()
{
  long *ptr;
  register int i;

  ptr = vec_long(20);

  for(i=0;i<20;i++)
    ptr[i] = i+1;

  for(i=0;i<20;i++)
    printf("ptr[%2d] = %2d\n", i, ptr[i]);

  free(ptr);
}
```

Box 7 - Utilizando a função vec_long.

No Box 8 apresentamos o protótipo da última função disponível na biblioteca *stdlib.h* para trabalhar com memória dinâmica, chamada de *realloc*.

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

Box 8 - Protótipo da função realloc.

Esta função tem o objetivo de modificar o tamanho da memória que foi previamente alocada e apontada por *ptr* para o tipo *size*. A função *realloc* retorna um ponteiro para o bloco de memória, pois ela poderá mover esse bloco para ampliar seu tamanho. Se *ptr* for NULL, *realloc* devolve um ponteiro para a memória alocada. Caso *size* seja zero, será liberada a memória apontada por *ptr*. O código do Box 9 ilustra uma aplicação do *realloc*, quando deseja-se realocar espaço para concatenar duas *strings*.

```
main()
{
  char *palloc;

  palloc = malloc(8);

  if(!palloc)
  {
    puts("Fatal error");
    system("pause");
    exit(1);
  }

  strcpy(palloc,"WR KITS");

  palloc = realloc(palloc,19); /* realoca para 19 bytes */

  if(!palloc)
  {
    puts("Fatal error");
    system("pause");
    exit(1);
  }

  strcat(palloc," Engenharia");

  printf(palloc);
  free(palloc);

}
```

Box 9 - Realocando memória.

Exercício proposto: Desenvolva o projeto de um leitor binário que apresente na tela os bytes organizados conforme Figura 1.

```
00 01 02 03 04 05 06 07 08 09
-- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- --
.
.
.
```

Figura 1 - Leitor de arquivos binários.

Esse vetor de bytes para exibição do conteúdo do arquivo binário, deverá ser alocado dinamicamente.

Vale destacar que existem diversas outras funções de alocação dinâmica de memória, estas implementadas na biblioteca *malloc.h*, tais como *_alloca*, *_ffree*, *_fmsize*, *_freect*. O aluno interessado poderá estuda-las em detalhes na obra C Completo e Total, do Herbert Schildt. Não abordaremos as mesmas pois as 4 funções estudadas presentes em *stdlib.h* já são mais que suficientes para 99% das aplicações envolvendo memória dinâmica.

Bibliografia:

DAMAS, Luís; Linguagem C, décima edição.

Disponível em: https://amzn.to/3nGdIbN

SCHILDT, Herbert; C Completo e Total, terceira edição.

Disponível em: https://amzn.to/3xxi3l8