Alocação dinâmica de memória

Esta página é um pequeno resumo sobre alocação dinâmica de memória, ou seja, alocação que ocorre durante a execução de um programa.

Malloc e free

A função `malloc` (abreviatura de *memory allocation*) da biblioteca padrão aloca um bloco de *bytes* consecutivos na memória do computador e devolve o endereço desse bloco. Eis um exemplo que recebe um caracter e imprime o caracter seguinte da tabela ISO 8859-1:

```
char *ptr;
ptr = (char *)malloc (1);
scanf ("%c", ptr);
printf ("%c\n", *ptr + 1);//não confundir com *(ptr+1)
free (ptr);
```

Eis outro exemplo bobo, que calcula a soma de dois números:

```
  int *ptr;
  int x;
  ptr = (int *)malloc (sizeof (int));
  scanf ("%d %d", ptr, &x);
  *ptr = *ptr + x;
  printf ("%d\n", *ptr);
  free (ptr);
```

A expressão `sizeof (int)` dá o número de *bytes* de um `int`. No meu computador, esse número é 4.

A função `malloc` devolve um ponteiro "genérico" — ou seja, do tipo **`void *`** — para um bloco de *bytes* consecutivos. No exemplo acima, esse ponteiro é automaticamente convertido em ponteiro-para-int uma vez que a variável `ptr` é do tipo `int *`.

A função `free` liberta a porção de memória alocada por `malloc`. O comando `free (ptr)` avisa o sistema de que o bloco de *bytes* apontado por `ptr` está livre. A próxima chamada de `malloc` poderá tomar posse desses *bytes*.

As funções `malloc` e `free` estão na biblioteca `stdlib`. Portanto, inclua

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

no início de qualquer programa que use `malloc` e `free`.

A memória não é infinita

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 = (int *)malloc (sizeof (int));
   if (ptr == NULL) {
      printf ("Socorro! malloc devolveu NULL!\n");
      exit(0);
   }
```

A codificação freqüente e repetida desse teste é cansativo para o programador e desvia a atenção. Deste modo poderemos usar a seguinte versão alternativa de `malloc`:

(Função implementada pelo programador!!!!)

```
void *Alocar (unsigned int nbytes)
{
   void *ptr;
   ptr = malloc (nbytes);
   if (ptr == NULL) {
      printf ("Socorro! malloc devolveu NULL!\n");
      exit (0);
   }
   return ptr;
}
```

A função **Alocar** *não está em nenhuma biblioteca* e é desconhecida fora dessas notas de aula. Ela é apenas uma abreviatura conveniente.

Exemplo 1

Eis como um vetor *(= array)* com `n` elementos inteiros pode ser alocado durante a execução de um programa:

```
   int *v;
   int n, i;

   scanf ("%d", &n);
   v = (int *)Alocar (n * sizeof (int));
   for (i = 0; i < n; i += 1)
      scanf ("%d", &v[i]);
   for (i = 0; i < n; i += 1)
      printf ("%d ", v[i]);
   free (v);
```

Do ponto de vista conceitual (mas apenas desse ponto de vista) o comando

```
v = Alocar (100 * sizeof (int));
```

tem o mesmo efeito que

```
int v[100];
```

Repare-se que quando se faz

```
   int v[100];
```

Não é necessário fazer-se free; pois não foi feita uma alocação com o malloc.