MINISTÈRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE ITAJUBÁ
Criada pela Lei nº 10.435 – 24/04/2002

Algoritmos e Estrutura de Dados I
Anotações de Aula
Profa Melise Maria Veiga de Paula

Alocação Dinâmica

Em C, existem duas estratégias para armazenamento de informações na memória. Uma delas é a utilização estática de variáveis (locais ou globais) que incluem os vetores, matrizes e os registros (struct) estáticos. Para este tipo de alocação, é necessário conhecer, de antemão, qual a quantidade de memória necessária à solução do problema.

Ao elaborar um código, o programador deve definir quais variáveis serão necessárias. Ao declarar um vetor ou uma matriz, o programador deve definir qual a dimensão destas estruturas de dados. Uma vez tendo compilado o programa, durante a execução, não é permitido alocar memória para novas variáveis ou alterar a dimensão das estruturas. Devido a este comportamento, esta estratégia é denominada alocação estática de memória, ou seja, a porção de memória alocada aos dados do seu programa não muda, permanece estática durante toda a execução.

A outra estratégia para manipular a memória é alocar espaço da memória dinamicamente. Com esta estratégia, durante a execução de um programa, o programador consegue alocar novas "porções" de memória ou liberar de acordo com o "tamanho" do seu problema.

Esta estratégia é requerida quando são necessárias quantidades de armazenamento variáveis. Por exemplo, uma lista de clientes dentro de um programa pode aumentar ou diminuir dinamicamente. Se um vetor estático for usado como mecanismo para manutenção desta lista, o número de clientes será limitado, ou ainda, pode-se desperdiçar espaço caso o número de clientes tenha sido superestimado. Se a alocação dinâmica for usada, a limitação que podemos ter será em função do espaço de memória disponível quando da execução do programa.

Vale lembrar que para um programa em C, a memória é dividida em três regiões: memória estática (variáveis globais e constantes), a pilha (variáveis locais) e o heap. A memória dinâmica é alocada no heap.

São basicamente duas as funções de alocação dinâmica de memória em C que vamos usar: malloc e free.

malloc- aloca n bytes e devolve o endereço do primeiro byte da memória alocado. O retorno da função deve ser atribuído a uma variável do tipo ponteiro. No Exemplo 1, foram alocados 1000 bytes de memória sendo que p aponta para o primeiro destes 1000 bytes.

Exemplo 1:

```
char *p;
p = malloc(1000);
```

Sempre que se alocar memória deve-se testar o valor devolvido por malloc (), antes de

usar o ponteiro, para garantir que não é nulo. Assim:

```
if (!p){
     printf ("sem memoria\n");
     exit (1);
}
```

A função malloc devolve um ponteiro para void (sem tipo) pois a memória alocada pode ser usada para armazenar diferentes tipos de dados. Logo, ao usar a função, o correto é "moldar" o valor retornado por malloc para o tipo do dado que se deseja armazenar nesse espaço, ou seja, o ponteiro deve ser "casted" (tipado, moldado) para o tipo desejado. No exemplo 1, como p é um ponteiro para char. A função malloc deve ser usada da seguinte forma:

```
char *p;
p = (char *) malloc (1000);
```

free- afunção free é a função simétrica de malloc, visto que ela devolve para o sistema uma porção de memória que foi alocada dinamicamente.

Veja que, no exemplo 1, supondo que um char ocupe 1 byte, o espaço alocado é suficiente para armazenar 1000 char. Neste caso, estamos supondo que o tamanho do tipo é conhecido.
Contudo, para evitar problemas, recomenda-se usar a função sizeof(tipo). Esta função retorna o tamanho de memória necessária para alocar uma variável de um determinado tipo. Com essa nova estratégia, a função malloc para alocar um char ficaria assim:

```
char *p;
p = (char *) malloc(sizeof(char));
```

A função malloc para alocar 1000 char ficaria assim:

```
char *p;
p = (char *) malloc(sizeof(char)*1000);
```

A seguir um exemplo:

```
/* Exemplo - Alocacao dinamica de memoria */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
main ()
{
  char *s;
  int t;
  printf ("\n\nQual o tamanho da string ? ");
  scanf("%d",&t);
  s = (char *) malloc(sizeof(char)*t);
  if (!s)
    {
      printf("Sem memoria!");
      exit(1);
    }
```

```
printf("Entre um string qualquer : ");
scanf("%s",s);
printf("\nAo contrario fica..... : ");
for(t = strlen(s) - 1; t >= 0; t--)
      printf(" %c",*(s+t)); // equivalente a printf(" %c",s[t]);
free(s);
printf("\n");
system("pause");}
```

Exercícios

1. Faça uma função que receba dois vetores de inteiros, com qualquer número de elementos cada. A função deve imprimir todos os valores comuns aos dois vetores. Ex: se v1={19, 5, 2, 6} e v2={5, 0, 9, 4, 18, 56}, deverá ser impresso somente o valor 5. O número de elementos de cada vetor deverá ser fornecido pelo usuário.