LABORATÓRIO DE PROGRAMAÇÃO I

Profa. Gisele Busichia Baioco
gisele@ft.unicamp.br

Algoritmos Estruturados e Linguagem de Programação Estruturada

Ponteiros em C
Parte III – Alocação Dinâmica

1 Introdução

Existem dois métodos principais por meio dos quais um programa em C pode alocar memória do computador para armazenar informações:

- **Alocação Estática:** obtida por meio de declaração de variáveis. Para cada variável declarada no programa, o compilador aloca uma porção de memória correspondente ao tipo de dado que irá armazenar (especificado na declaração da variável). Inclusive, no caso do uso de vetores, matrizes e *strings*, é necessário especificar no programa-fonte o número de elementos mesmo que algumas posições possam não ser utilizadas durante a execução do programa. Esse tipo de alocação de memória é denominado **Alocação Estática;**
- **Alocação Dinâmica:** obtida por meio do uso de ponteiros, juntamente com funções de biblioteca que alocam memória durante a execução do programa. A quantia de memória alocada deve ser especificada, mas pode ser determinada em tempo de execução de acordo com a necessidade. Por exemplo, é possível solicitar ao usuário informações que determinem a quantidade de memória necessária em um dado momento da execução de um programa.

A alocação dinâmica é útil quando não se sabe com antecedência (antes da execução do programa) quantos dados serão utilizados.

2 As funções malloc() e free()

As funções **malloc()** e **free()** compõem o sistema de alocação dinâmica da linguagem C, fazendo parte de sua biblioteca. Na realidade existem outras funções de alocação dinâmica na biblioteca de C que consistem em variações de **malloc()** e **free()**. Para utilizar as funções **malloc()** e **free()** em um programa deve-se utilizar a biblioteca **stdlib.h.**

Para alocar memória deve-se utilizar a função **malloc()**. Após a utilização de uma porção de memória alocada por **malloc()**, deve-se utilizar a função **free()** para liberar a porção de memória que não será mais utilizada.

A declaração de **malloc()** é a seguinte:

```
void *malloc(int número-de-bytes);
```

A função **malloc()** retorna um ponteiro do tipo void, que significa que deve-se usar um *cast* (conversão de tipo) explícito no momento de atribuir o ponteiro retornado por **malloc()** a um ponteiro do tipo de dado desejado. Se **malloc()** tiver sucesso (conseguir alocar memória), devolverá um ponteiro para o primeiro byte da região de memória que foi alocada; se não houver memória suficiente disponível para satisfazer o pedido de **malloc()**, esta devolverá

um ponteiro nulo (inválido). Desse modo, antes de usar o ponteiro que **malloc()** retorna, deve-se assegurar que seu pedido de alocação foi bem sucedido testando o valor retornado contra zero (ponteiro nulo).

Juntamente com **malloc()**, pode-se usar a função de biblioteca **sizeof()** para determinar o número exato de bytes que cada tipo de dado precisa a fim de alocar a memória desejada.

A declaração de **free()** é a seguinte:

```
free(void *p);
```

Exemplo: o programa seguinte alocará memória suficiente para 40 inteiros, imprimirá seus valores e depois devolverá essa porção de memória ao sistema.

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

main()
{
   int *p, i;
   p = (int *)malloc(40 * sizeof(int));
   if (!p) /* ou p == 0 testa se a alocação foi bem sucedida */
      printf("memória insuficiente\n");
   else {
      for (i = 0; i < 40; i++)
         p[i] = i;   /* ou *(p + i) = i */
      for (i = 0; i < 40; i++)
         printf("%d ", p[i]);  /* ou *(p + i) */
      free(p);
   }
}
```

Deve-se lembrar sempre que antes de usar um ponteiro retornado por **malloc()** é necessário assegurar de que a alocação foi bem-sucedida testando o valor retornado em relação a zero. No exemplo anterior caso **if (!p)** ou **if (p == 0)** for verdadeiro significa que a alocação não foi bem-sucedida.

3 Exemplos de Programas

1) Deseja-se armazenar o número de matrícula de N alunos de uma Universidade. Também é necessário armazenar as notas dos N alunos. Utilizando alocação dinâmica, fazer um programa C que:

- leia a quantidade N de alunos;
- leia as informações dos N alunos armazenando os dados separadamente;
- leia um determinado número de matrícula e determine a nota do aluno correspondente, repetindo o processo até que o usuário deseje parar.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>

main()
{
   /* Declaração de variáveis */
   int N, *X, i, mat, achou;
   float *Y;
   char resp;
   /* Entrada de dados */
```

```
   printf(“Digite a quantidade de alunos: ”);
   scanf(“%d”, &N);
   X = (int *)malloc(N * sizeof(int));  /* aloca memória para armazenar a
                                           matrícula dos N alunos */
   Y = (float *)malloc(N * sizeof(float));  /* aloca memória para armazenar
                                               a nota dos N alunos */
   if (!X || !Y)    /* testa se a alocação foi bem sucedida */
      printf("memória insuficiente\n");
   else {
      for(i = 0; i < N; i++)
      {
         printf(“Digite o número da matrícula do aluno: ”);
         scanf(“%d”, &X[i]);
         printf(“Digite a nota do aluno: ”);
         scanf(“%f”, &Y[i]);
      }
      /* Processo e Saída */
      do {
         /* Lê a matrícula do aluno para consulta */
         printf(“\nEntre com a matrícula do aluno para consulta: ”);
         scanf(“%d”, &mat);
         /* Percorre o vetor X, procurando pela matricula */
         achou = 0;   /* não achou */
         i = 0;
         while (i < N && achou == 0)
         {
            if (X[i] == mat)
            {
               achou = 1;  /* achou */
               printf(“Nota = %f”, Y[i]);
            }
            i = i + 1;
         }
         /* Verifica se a matrícula não foi encontrada */
         if (achou == 0)   /* não achou */
            printf(“Matricula não encontrada”);
         /* Permite que o usuário faça mais uma consulta se desejar */
         printf(“\nDeseja consultar mais um aluno? (S/N)”);
         resp = getche();
      } while (toupper(resp) == ‘S’);
      free(X);
      free(Y);
   }
}
```

2) Refazer programa C anterior armazenando os dados dos N alunos em uma estrutura contendo como membros o número da matrícula e a nota do aluno.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>

main()
{
   /* Declaração de variáveis */
   struct aluno {
      int matricula;
      float nota;
      } *A;
   int N, i, mat, achou;
```

```
   char resp;
   /* Entrada de dados */
   printf(“Digite a quantidade de alunos: ”);
   scanf(“%d”, &N);
   A = (struct aluno *)malloc(N * sizeof(struct aluno));/* aloca memória */
   if (!A)    /* testa se a alocação foi bem sucedida */
      printf("memória insuficiente\n");
   else {
      for(i = 0; i < N; i++)
      {
         printf(“Digite o número da matrícula do aluno: ”);
         scanf(“%d”, &A[i].matricula);
         printf(“Digite a nota do aluno: ”);
         scanf(“%f”, &A[i].nota);
      }
      /* Processo e Saída */
      do {
         /* Lê a matrícula do aluno para consulta */
         printf(“\nEntre com a matrícula do aluno para consulta: ”);
         scanf(“%d”, &mat);
         /* Percorre A, procurando pela matricula */
         achou = 0;   /* não achou */
         i = 0;
         while (i < N && achou == 0)
         {
            if (A[i].matricula == mat)
            {
               achou = 1;  /* achou */
               printf(“Nota = %f”, A[i].nota);
            }
            i = i + 1;
         }
         /* Verifica se a matrícula não foi encontrada */
         if (achou == 0)   /* não achou */
            printf(“Matricula não encontrada”);
         /* Permite que o usuário faça mais uma consulta se desejar */
         printf(“\nDeseja consultar mais um aluno? (S/N)”);
         resp = getche();
      } while (toupper(resp) == ‘S’);
      free(A);
   }
}
```