Estrutura de Dados

Ponteiros e Alocação Dinâmica

Por Juliano P. Menzen

Sumário

- Ponteiros;
- Alocação Dinâmica;
- Atividades;

Ponteiros ou Apontadores

- Fornecem um mecanismo poderoso, flexível e eficiente de acesso a variáveis (memória);
- Cada objeto (variável ou função) que reside no computador possui um endereço de memória.
- Um apontador é uma variável que armazena um endereço de outra variável;

Endereço físico	Endereço lógico
1000	x
1002	a
1003	v[0]
1005	v[1]
1007	v[2]
1009	v[3]

Endereço físico	Endereço lógico	Conteúdo
1000	x	10
1002	y	20
1004	px	1000

Ponteiros ou Apontadores

- Utilizações:
  - Fornecem maneiras com as quais as funções podem realmente modificar os argumentos que recebem;
  - Para passar estruturas homogêneas e heterogêneas mais convenientemente de uma função para outra, possibilitando a manipulação mais facilmente através da movimentação de apontadores para elas;
  - Suportam as rotinas de alocação dinâmica em C;
  - Para criar estruturas de dados mais complexas, como listas encadeadas e árvores;

Ponteiros ou Apontadores

- Declarando um ponteiro:

  <Tipo de Dado> * <Nome da Variável>;

- **Tipo da Dado :** é o identificador de tipo de informação, qualquer tipo válido em C, para qual a variável vai apontar.
- **Nome da Variável:** Em geral, pode-se dar um nome qualquer a uma variável válida em C.
- Exemplo:
  - char * p3;

Ponteiros ou Apontadores

- Operações com apontadores:
- O operador **&** é um operador unário que retorna o endereço de memória do seu operando.
- O operador ***** é um operador que retorna o conteúdo da variável localizada no endereço (ponteiro) do operando, ou seja, devolve o conteúdo da variável apontada pelo operando.
- O indicador de impressão **%p** indica que um endereço de um ponteiro em hexadecimal deve ser impresso, usando notação hexadecimal.

Ponteiros ou Apontadores

- Exemplos de operações:
  - int x=10; // declara uma variável do tipo inteiro
  - int * p1; // declara uma variável do tipo ponteiro para inteiro
  - p1 = &x; // armazena em p1 o endereço de memória da variável x
  - printf("\n O valor de p1: %p", p1); // imprime o endereço de memoria em hexa
  - printf("\n O conteudo de p1: %d", *p1); // imprime o conteúdo do ponteiro

- Quando é realizada uma atribuição de uma variável para um ponteiro a quantidade de informação copiada é relativa ao tamanho do tipo de dado da variável (tipo base);

Ponteiros ou Apontadores

- Expressões com apontadores:
  - Podem ser utilizadas da mesma forma que as expressões em variáveis, mas sempre estarão sendo executadas em endereços de memória;
  - Atribuindo endereço de uma variável para um ponteiro:
    - ***<Nome do Ponteiro> = & <Nome da Variável>;***
  - ***Atribuindo o conteúdo de um ponteiro para uma variável:***
    - ***<Nome da Variável > = * <Nome do Ponteiro >;***
  - ***Para atribuir um ponteiro a outro:***
    - **<Nome do Ponteiro 1 > = <Nome do Ponteiro 2 >;**
  - **Para atribuir o conteúdo de um ponteiro para outro:**
    - ***<Nome do Ponteiro 1 > = *<Nome do Ponteiro2 >;**

Ponteiros ou Apontadores

- Incremente e Decremento de ponteiros:
  - ***<Nome do Ponteiro>++; //incrementa***
  - ***OU***
  - ***<Nome do Ponteiro> - - ; //decrementa***
  - ***Este comando incrementa o ponteiro para o próximo endereço de memória de mesmo tipo;***
    - **(*<Nome do Ponteiro>)++;**
    - **//incrementa o conteúdo da variável apontada pelo ponteiro**
    - **ou**
    - **(*<Nome do Ponteiro>) - - ;**
    - **//decrementa o conteúdo da variável apontada pelo ponteiro**
  - Este comando incrementa o valor do ponteiro;
  - OBS: os operadores de incremento e decremento tem prioridade sobre os operadores de ponteiros (* e &);

Ponteiros ou Apontadores

- Apontadores para vetores
  - Ao declarar um vetor também estamos manipulando ponteiros;
    - **int Num[10];**
  - Num é um apontador para inteiro;
  - O valor da variável Num é igual ao endereço do primeiro elemento do vetor, ou seja, igual a &Num[0];
  - Como Num é um ponteiro, referencia-se os valores do vetor indexando através de operações com ponteiros.
    - **vetor [índice]** <=> ***(vetor + índice)**
  - Para referenciar o endereço de um elemento do vetor:
    - **&vetor [índice]** <=> **(vetor + índice)**

Ponteiros ou Apontadores

- Apontadores para vetores
  - Exemplo:
    - int i, vetor[10], *pvetor=NULL;
    - pvetor = vetor;
    - for(i=0;i<10;i++)
      - *(pvetor+i) = i+1;
    - for(i=0;i<10;i++)
      - printf("\n O conteudo de pvetor[%d] = %d" , i, *(pvetor+i));

Ponteiros ou Apontadores

- Apontadores para Estruturas (struct)
  - Pode-se utilizar ponteiros para referenciar structs assim como qualquer outro tipo de variável;
  - Sintaxe de declaração:
    - **struct Nome_Estrutura * NomePonteiro;**
      - Ou
    - **Nome_Estrutura * NomePonteiro; //se a estrutura já estiver definida**
  - Exitem duas formas para acessar os valores da struct:
    - **(*NomePonteiro) . Membro //deve-se usar parênteses por conta da precedência do . Pelo ***
      - Ou
    - **NomePonteiro -> membro**

Ponteiros ou Apontadores

- Apontadores para Apontadores
  - Permite guardar em um apontador o endereço de memória de outra apontador;
    - **<Tipo Base> ** <NomePonteiro>;**
  - Algumas considerações:
    - ****NomePonteiro** é o conteúdo final da variável apontada;
    - ***NomePonteiro** é o conteúdo do ponteiro intermediário.
  - Exemplo:

```
main (void )
{
        float fpi = 3.1415,  *pf,  **ppf;
        system("cls");
        pf = &fpi;                  // pf armazena o endereço de fpi
        ppf = &pf;                  // ppf armazena o endereço de pf
        printf("%f", **ppf);        // Imprime o valor de fpi
        printf("\n%f", *pf);        // Também imprime o valor de fpi
        getch( );
}
```

Alocação Dinâmica

- A alocação dinâmica permite ao programador alocar memória para variáveis enquanto o programa está sendo executado;
- Exemplo:
  - Pode-se aumentar o tamanho de uma string dinâmicamente;
  - Pode-se aumentar o tamanho de um vetor de inteiros após a sua declaração;

Alocação Dinâmica

- Sintaxe:
  - ***void * calloc (Num_Elementos, Tamanho_Elemento);***
- Onde:
  - **Num_Elementos :** é um número inteiro sem sinal que define quantidade de elementos para qual serão alocados memória.
  - **Tamanho_Elemento:** é um número inteiro sem sinal que define o número de bytes de cada elemento. Para saber o número de bytes de um tipo de dado, utiliza-se a função **sizeof(tipo_de_dado)**. Esta função retorna o número de bytes do tipo_de_dado passado como argumento.
  - **void *:** este é o tipo de retorno da função calloc, um ponteiro do tipo void. Este ponteiro pode ser atribuído a qualquer tipo ponteiro.

Alocação Dinâmica

- Retorno da função:
  - o **ENDEREÇO** para o primeiro byte alocado.
- **ou**
  - o valor **NULL**, quando não foi possível alocar memória.
- Chamada:
  - **Ponteiro = (tipo *) calloc (Num_Elementos, Tamanho_Elemento);**

- Para liberar a memória alocada, após o uso deve-se usar a função "***free***";
  - **free(ponteiro);**

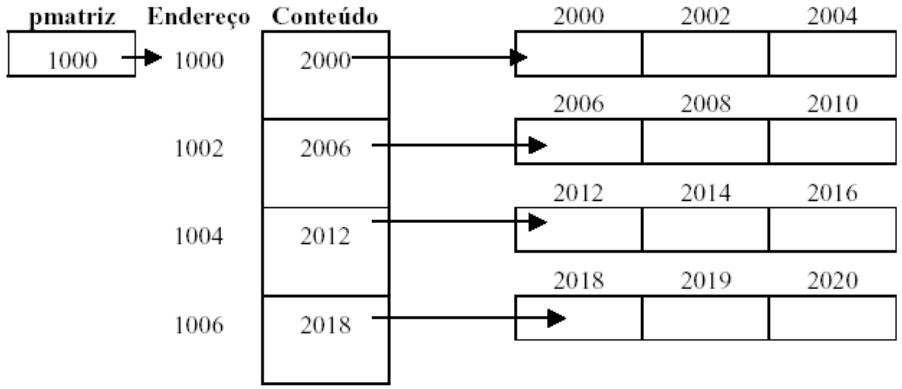
Alocação Dinâmica

- Exemplo de alocação para vetor:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h> // Para usar calloc( ), free( ) e exit( )

int *Aloca_Vetor(int tam);

main (void )
{
              int n, *pvetor, i;
              system(“cls”);
              printf(“Digite o número de elementos do vetor:”);
              scanf(“%d”, &n);
              pvetor = Aloca_Vetor (n);     // pvetor pode ser tratado como um vetor com n posições

              for (i=0; i < n; i++) {
                            pvetor[i] = i*i;          // *(pvetor + i) = i*i ;
                            printf("pvetor[%d] = %d\n", i, pvetor[i]);
              }
              free (pvetor);
              getch();
}

int *Aloca_Vetor(int tam)
{
              int * v;      // ponteiro para o vetor
              v = (int *) calloc(tam, sizeof(int) );   //Aloca tam números inteiros; v pode agora ser tratado como um vetor com tam posições.
              if (!v)                    //(v == NULL)
              {
                            printf ("** Erro: Memória Insuficiente **");
                            exit(1);
              }
              return v;     // retorna o ponteiro para o vetor
}
```

Alocação Dinâmica

- Alocação Dinâmica de Matrizes
  - A alocação dinâmica pode ser feita para matrizes da mesma forma que para vetores;
  - Porem, deve-se ter um ponteiro apontando para outro ponteiro que aponta para o valor final;
  - Esta característica é chamada de ***indireção múltipla***;
  - Através deste conceito de indireção múltipla, pode-se alocar dinamicamente matrizes de ***N*** dimensões;

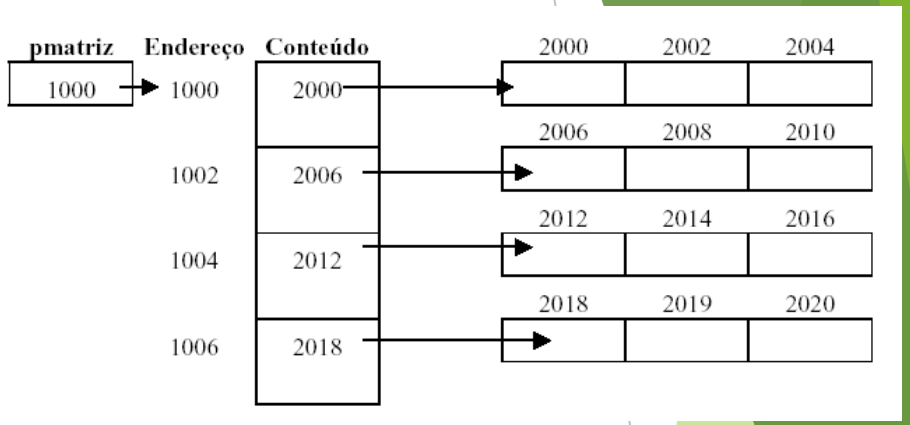


Alocação Dinâmica

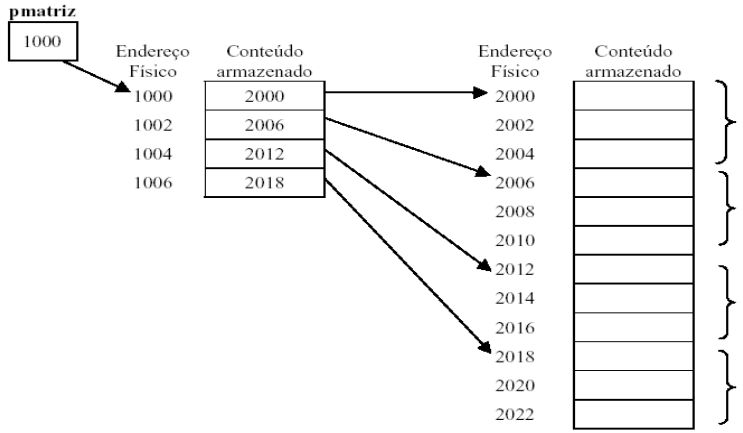
- Formas de uso:
  - Para acessar os valores de uma posição da matriz:
    - **matriz [índice1][índice2] ⇔. *(*(matriz + índice1) + índice2)**
  - Para acessar os endereços de memória:
    - **&matriz [índice1][índice2] ⇔ *(matriz + índice1) + índice2)**

Alocação Dinâmica

- Exemplos:
  - Exemplo através de alocação de um vetor de ponteiros:
    - Vide arquivo: exemplo1.cpp;



  - Exemplo através de alocação de um vetor de ponteiros com apenas um ponteiro:
    - Vide arquivo: exemplo2.cpp;



Alocação Dinâmica

- Cuidados:
  - Nunca utilizar um ponteiro que não foi inicializado;
  - Não invadir o espaço de outras variáveis;
  - Cuidado em struct que possuem propriedades que são ponteiros:
    - Neste caso pode-se atribuir uma mesma variável apontadora para dois elementos que deveriam ser diferentes;

Atividades

- 1) Ler um vetor de inteiros e calcular seu valor máximo. O programa deve solicitar o tamanho do vetor e deve retornar o maior valor atribuído a alguma posição do vetor;