Instituto Federal Catarinense - Câmpus Videira

Bacharel em Ciência da Computação

Guilherme Naderer Chaves

Relatório Trabalho Final

Problema: Processamento de Arrays

Julho - 2021

Primeiramente, este é um problema que nos traz um elemento especial, as matrizes, que são nada mais nada menos que um vetor em que cada posição do vetor tem um vetor dentro, e estes vetores por sua vez tem suas posições e etc…, desta maneira para resolver este problema, se fez necessário uso de ponteiros e alocação dinâmica de memória para conseguir lidar com matrizes e módulos ao mesmo tempo, pois é muito mais fácil de usar no código, ao invés de tentar utilizar-se de matrizes e módulos sem ponteiro e alocação dinâmica.

Então com os módulos podemos além de separar o código e assim deixando ele mais organizado ainda conseguimos, de tal forma não repetir as mesmas linhas (ou parecidas) de código quando é requisitado fazer um processo igual ou que seja bem parecido, normalmente isso nunca acontece quando começa a programar o código, ou seja, você resolveu um problema igual e/ou parecido anteriormente e agora tem outro, assim que se encontrarmos em esta situação, podemos ou devemos usar modularização, principalmente quando envolve pessoas que não estavam junto na programação do código e/ou vão analisar o mesmo.

Logo, com os módulos a nossa parte “principal” fica mais clara para entendimento, onde fazemos algumas perguntas para o usuário do programa e pegamos as respostas que precisamos para podermos chamar as funções, os módulos, passando a informações por parâmetro.

Posso agora falar e demonstrar um pouco do código que eu fiz.

Vou começar com o coração do código, o “main”:

```c
int main(void) {

  //setando as variáveis
  int tamanho=0;
  int ** matrizOrigem;
  int ** matrizFixa;
  int ** matrizResultante;

  //entrada de dados
  printf("\n");
  printf("Digite o tamanho da matriz origem: ");
  scanf("%d", &tamanho);
  printf("\n");

  int tamanhoResultante=tamanho-2;

  //chamando as funções
  matrizOrigem = alocaMatriz(tamanho);//alocando espaço de memória
  matrizFixa = alocaMatriz(3);//alocando espaço de memória
  matrizResultante = alocaMatriz(tamanhoResultante);//alocando espaço de memória
  preencheMatriz(matrizOrigem,tamanho, 1);//entrada de dados - preenchimento da matriz
  preencheMatriz(matrizFixa, 3, 2);//entrada de dados - preenchimento da matriz
  calculaMatriz(matrizOrigem, matrizFixa, matrizResultante, tamanhoResultante);//processamento de dados - calculando os resultados para a matriz que recebe os valores
  printaMatriz(matrizOrigem, tamanho, matrizFixa, 3, matrizResultante, tamanhoResultante); //saída de dados - mostrando ao usuário como ficou cada matriz
  liberaMemoria(matrizOrigem, tamanho);//liberando espaço de memória
  liberaMemoria(matrizFixa, 3);//liberando espaço de memória
  liberaMemoria(matrizResultante, tamanhoResultante);//liberando espaço de memória
}
```

Aqui acima temos o que comentei anteriormente, as perguntas para o usuário para podermos chamar as funções/módulos e passar essas informações por parâmetro.

Agora os módulos, por ordem que estão aparecendo no main:

```
int ** alocaMatriz(int tam){
  int ** matriz;
  matriz = (int **) calloc(tam, sizeof(int *));
  for(int i=0; i<tam; i++){
    matriz[i]=(int*) calloc(tam, sizeof(int));
  }
  return matriz;
}
```

Acima temos o módulo chamado “alocaMatriz”, ele é atarefado de reservar/alocar espaço de memória que será utilizado posteriormente no decorrer do código.

Abaixo está o módulo com nome de “preencheMatriz”, este por sua vez é responsável por perguntar ao usuário, desta forma coletando o dado que o mesmo der como entrada e então inserir nas matrizes as informações.

```
void preencheMatriz(int **matriz,int tam, int qual){
  printf("\n");
  int preenche=0;
  if(qual==1){
    for(int linha=0; linha<tam; linha++){
      for(int coluna=0; coluna<tam; coluna++){
        printf("Digite um número inteiro para preencher a Matriz Origem: ");
        scanf("%d", &preenche);
        matriz[linha][coluna]=preenche;
      }
    }
  }
  else if(qual==2){
    for(int linha=0; linha<tam; linha++){
      for(int coluna=0; coluna<tam; coluna++){
        printf("Digite um número inteiro para preencher a Matriz Fixa: ");
        scanf("%d", &preenche);
        matriz[linha][coluna]=preenche;
      }
    }
  }
}
```

Agora o módulo abaixo é chamado de “calculaMatriz”, utilizado somente para a matriz Resultante, ele é responsável pelo processamento dos dados anteriormente fornecidos e, com eles, fazer os cálculos para preencher com o resultado destes a matriz Resultante. Para estes processos acontecerem é necessário a passagem das matrizes já preenchidas, e fiz isso passando elas por parâmetro.

```
void calculaMatriz(int ** matrizOrigem, int ** matrizFixa, int ** matrizResultante, int tamR){
  int soma=0;
  for(int li=1; li<=tamR; li++){
    for(int co=1; co<=tamR; co++){
      soma+= (matrizOrigem[li-1][co-1] * matrizFixa[0][0]) + (matrizOrigem[li-1][co] * matrizFixa[0][1]) + (matrizOrigem[li-1][co+1] * matrizFixa[0][2]);
      soma+= (matrizOrigem[li][co-1] * matrizFixa[1][0]) + (matrizOrigem[li][co] * matrizFixa[1][1]) + (matrizOrigem[li][co+1] * matrizFixa[1][2]);
      soma+= (matrizOrigem[li+1][co-1] * matrizFixa[2][0]) + (matrizOrigem[li+1][co] * matrizFixa[2][1]) + (matrizOrigem[li+1][co+1] * matrizFixa[2][2]);
      matrizResultante[li-1][co-1]=soma;
      soma=0;
    }
  }
}
```

Também temos o módulo denominado de “printaMatriz”, o qual tem como função mostrar ao usuário como as matrizes ficaram preenchidas, e para isto, passei as matrizes e seus respectivos tamanhos por parâmetro. Confira abaixo:

```
void printaMatriz(int ** matriz1, int tam1, int ** matriz2, int tam2, int ** matriz3, int tam3){
  printf("\n");
  printf("A Matriz Origem ficou assim: ");
  printf("\n\n");
  for(int linha=0; linha<tam1; linha++){
    for(int coluna=0; coluna<tam1; coluna++){
      printf("| %d |", matriz1[linha][coluna]);
    }
    printf("\n");
  }
  printf("\n");
  printf("A Matriz Fixa ficou assim: ");
  printf("\n\n");
  for(int linha=0; linha<tam2; linha++){
    for(int coluna=0; coluna<tam2; coluna++){
      printf("| %d |", matriz2[linha][coluna]);
    }
    printf("\n");
  }
  printf("\n");
  printf("A Matriz Resultante ficou assim: ");
  printf("\n\n");
  for(int linha=0; linha<tam3; linha++){
    for(int coluna=0; coluna<tam3; coluna++){
      printf("| %d |", matriz3[linha][coluna]);
    }
    printf("\n");
  }
}
```

E por último mas não menos importante, abaixo temos o módulo chamado “liberaMemoria”, onde o espaço e as informações guardadas são liberadas, assim liberando memória.

```
void liberaMemoria(int ** matriz, int tam){
  for(int i=0; i<tam; i++){
        free(matriz[i]);
  }
  free(matriz);
}
```