PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Programa de Graduação em Ciência da Computação

Henrique Oliveira da Cunha Franco
Pedro Henrique Gaioso de Oliveira

RELATÓRIO DE CÓDIGOS - INVESTIGAÇÃO DE PONTEIROS

Estudo aprofundado a respeito da aplicação e uso de ponteiros na linguagem de programação C

Belo Horizonte
2023

Henrique Oliveira da Cunha Franco
Pedro Henrique Gaioso de Oliveira

RELATÓRIO DE CÓDIGOS - INVESTIGAÇÃO DE PONTEIROS
Estudo aprofundado a respeito da aplicação e uso de ponteiros na linguagem de programação C

Relatório Técnico apresentada ao Programa de Graduação em Ciência da Computação da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Diploma em Ciência da Computação.

Orientador: Gustavo Luis Soares

Área de concentração: Estudo aprofundado de ponteiros

Belo Horizonte
2023

Henrique Oliveira da Cunha Franco
Pedro Henrique Gaioso de Oliveira

RELATÓRIO DE CÓDIGOS - INVESTIGAÇÃO DE PONTEIROS
Estudo aprofundado a respeito da aplicação e uso de ponteiros na linguagem de programação C

Relatório Técnico apresentado ao Programa de Graduação em Ciência da Computação da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Diploma em Ciência da Computação.

Área de concentração: Estudo aprofundado de ponteiros

Henrique Oliveira da Cunha Franco(Aluno) – PUC Minas

Pedro Henrique Gaioso de Oliveira(Aluno) – PUC Minas

Gustavo Luis Soares(Orientador) – PUC Minas

Belo Horizonte, 17 de Maio de 2023

RESUMO

A utilização de ponteiros em linguagem C é uma das características que tornam a linguagem tão flexível e poderosa. Ela por sua vez, se trata de variáveis que armazenam o endereço de memória de outras variáveis. Dizemos que um ponteiro "aponta" para uma varíável quando contém o endereço da mesma. Os ponteiros podem apontar para qualquer tipo de variável. Portanto temos ponteiros para int, float, double, etc. Ponteiros são muito úteis quando uma variável tem que ser acessada em diferentes partes de um programa. Neste caso, o código pode ter vários ponteiros espalhados por diversas partes do programa, "apontando" para a variável que contém o dado desejado. Caso este dado seja alterado, não há problema algum, pois todas as partes do programa tem um ponteiro que aponta para o endereço onde reside o dado atualizado.

Existem várias situações onde ponteiros são úteis, por exemplo:

- Alocação dinâmica de memória
- Manipulação de arrays
- Para retornar mais de um valor em uma função
- Referência para listas, pilhas, árvores e grafos

Palavras-chave: Modelo LaTeX. Relatório, defesa. Ponteiros.

ABSTRACT

The use of pointers in the C language is one of the features that make the language so flexible and powerful. They, in turn, are variables that store the memory address of other variables. We say that a pointer "points" to a variable when it contains its address. Pointers can point to any type of variable. So we have pointers to int, float, double, etc. Pointers are very useful when a variable has to be accessed in different parts of a program. In this case, the code can have several pointers transmitted by different parts of the program, "pointing" to a variable that contains the desired data. If this data is changed, there is no problem, as all parts of the program have a pointer that points to the address where the updated data resides.

There are a number of situations where pointers are useful, for example:

- Dynamic Memory Allocation
- Array manipulation
- Returning more than one value in a function
- Reference for lists, stacks, trees and graphs

Keysords: Template LaTeX. Technical Report. Pointers.

SUMÁRIO

1 INTRODUÇÃO **6**
1.1 Endereços 6
1.2 Ponteiros 7
1.2.1 Tipos de Ponteiros 7
1.2.2 Exemplo 7

2 DESENVOLVIMENTO **8**
2.1 Exercício 1 9
2.2 Questão 2 10
2.3 Questão 3 11
2.4 Questão 4 12
2.5 Questão 5 13
2.6 Questão 6 14
2.7 Questão 7 15
2.8 Questão 8 16
2.9 Questão 9 17
2.10 Questão 10 18
2.11 Questão 11 19
2.12 Questão 12 20
2.13 Questão 13 21
2.14 Questão 14 22
2.15 Questão 15 23
2.16 Questão 16 24
2.17 Questão 17 25
2.18 Questão 18 26
2.19 Questão 19 27
2.20 Questão 20 28
2.21 Questão 21 29
2.22 Questão 22 30

3 CONCLUSÃO **31**

REFERÊNCIAS **33**

1 INTRODUÇÃO

Esse relatório tem como objetivo fundamental o estabelecimento do contato de alunos com as normas da ABNT, bem como o entendimento por meio da manipulação e compreensão do tipo de variável Ponteiro, na linguagem C. Esse relatório é o meio de defesa e apresentação de 22 códigos feitos para resolução de exercícios simples realizados com o intuito de colocar o uso de ponteiros em prática. Abaixo lhe apresento uma curta introdução ao conceito de ponteiros nessa linguagem, a fim de contextualizar a prática de tal aspecto dessa linguagem.

1.1 Endereços

A memória RAM (=*random access memory*) de qualquer computador é uma **sequência** de **bytes**. A posição (0, 1, 2, 3, etc.) que um byte ocupa na sequência é o *endereço* (= *address*) do byte. (É como o endereço de uma casa em uma longa rua que tem casas de um lado só.) Se e é o endereço de um byte então e + 1 é o endereço do byte seguinte.

Cada variável de um programa ocupa um certo número de bytes consecutivos na memória do computador. Uma variável do tipo **char** ocupa 1 byte. Uma variável do tipo **int** ocupa 4 bytes e um **double** ocupa 8 bytes em muitos computadores. O número exato de bytes de uma variável é dado pelo operador *sizeof*. A expressão sizeof (char), por exemplo, vale 1 em todos os computadores e a expressão sizeof (int) vale 4 em muitos computadores.

Cada variável (em particular, cada registro e cada vetor) na memória tem um *endereço*. Na maioria dos computadores, o endereço de uma variável é o endereço do seu primeiro byte. Por exemplo, depois das declarações

```
char c;
int i;
struct {
   int x, y;
} ponto;
int v[4];
```

as variáveis poderiam ter os seguintes endereços (o exemplo é fictício):

```
c        89421
i        89422
ponto    89426
v[0]     89434
v[1]     89438
v[2]     89442
```

O endereço de uma variável é dado pelo operador **&**. Assim, se **i** é uma variável então &i é o seu endereço. (Não confunda esse uso de "&"com o operador lógico *and*, representado por "&&"em C.) No exemplo acima, &i vale *89422* e &v[3] vale *89446*.

Outro exemplo: o segundo argumento da função de biblioteca **scanf** é o endereço da variável que deve receber o valor lido do teclado:

```
int i;
scanf ("%d", &i);
```

1.2 Ponteiros

Um *ponteiro* é um tipo especial de variável que armazena um endereço. Um ponteiro pode ter o valor

NULL

que é um endereço "inválido". A **macro** NULL está definida na **interface stdlib.h** e seu valor é 0 (zero) na maioria dos computadores.

Se um ponteiro p armazena o endereço de uma variável i, podemos dizer "p aponta para i"ou "p é o endereço de i". (Em termos um pouco mais abstratos, diz-se que p é uma *referência* à variável i.)
Se um ponteiro p tem valor diferente de NULL então

***p**

é o *valor* da variável apontada por p. (Não confunda esse operador "*"com o operador de multiplicação!) Por exemplo, se i é uma variável e p vale &i então dizer "*p"é o mesmo que dizer "i".

A seguinte figura dá um exemplo. Do lado esquerdo, um ponteiro p, armazenado no endereço *60001*, contém o endereço de um inteiro. O lado direito dá uma representação esquemática da situação:

1.2.1 Tipos de Ponteiros

Há vários tipos de ponteiros: ponteiros para bytes, ponteiros para inteiros, ponteiros para ponteiros para inteiros, ponteiros para **registros**, etc. O computador precisa saber de que tipo de ponteiro você está falando. Para declarar um ponteiro p para um inteiro, escreva

```
int *p;
```

(Há quem prefira escrever **int* p**.) Para declarar um ponteiro p para um registro reg, diga

```
struct reg *p;
```

Um ponteiro r para um ponteiro que apontará um inteiro (como no caso de uma **matriz de inteiros**) é declarado assim:

```
int **r;
```

1.2.2 Exemplo

Suponha que a, b e c são variáveis inteiras e veja um jeito elaborado de fazer "c = a + b":

```
int *p; // p = ponteiro para um inteiro
int *q;
p = &a; // valor de p = o endereco de a
q = &b; // q aponta para b
c = *p + *q;
```

2 DESENVOLVIMENTO

Aqui será iniciada a apresentação e dissertação sobre os códigos apresentados que dizem respeito à manipulação e aprendizado sobre Ponteiros. A seguir, estará a função **main** do arquivo .c principal.

inputencoding

```
int main(void) {
  int num;
  printf("Escolha o exercicio desejado atraves"
  "da entrada de um numero de 1 a 22 ao terminal."
  "Tabela de Exercicios:"
  "Apresentando a declaracao basica de ponteiros                    [1]"
  "Apresentacao de interacoes entre ponteiros                       [2]"
  "Demonstracao do uso dos operadores & e *                         [3]"
  "Adicao de dois numeros utilizando ponteiros                      [4]"
  "Adicao de dois numeros a partir da chamada de variaveis          [5]"
  "Encontre o maior valor entre dois numeros                        [6]"
  "Armazenar e fazer a chamada de elementos contidos em um vetor    [7]"
  "Geracao de permutacoes de uma dada string                        [8]"
  "Busca do maior elemento atraves da utilizacao de Memoria Dinamica [9]");
  scanf("%d", &num);
}
```

Por motivos puramente didáticos, não foi incluída a seção do código com o elemento switch(case). Assuma que a inserção e confirmação do número digitado pelo usuário no terminal irá equivaler ao número ao lado de cada exercício, respeitando a tabela apresentada no código.
Agora, lhe apresento uma série de exercícios, separados por subseções, com uma breve descrição de cada código.

2.1 Exercício 1

inputencoding

```
int ex1() {
  srand(time(NULL));
  double m = rand() % 100, x, y, *z = &m;

  printf("Exercicio 1 - Apresentando a declaracao basica de ponteiros:\n");
  printf("-----------------------------------------------------------\n");
  printf("Digamos que m = %.lf, x e y sao duas variaveis do tipo int e *z "
         "tambem é uma variavel do tipo int.\n\n", m);
  printf("z armazena o endereco de    [m] = %p\n", z);
        // z é um ponteiro, ãento %p retorna o endereço
  printf("\n*z armazena o valor de     [m] = %.lf\n", *z);
  printf("\n&m = endereco de           [m] = %p\n", &m);
        // &m retorna o endereco da variavel m
        // %p serve para especificar que o retorno será um endereço
  printf("\n&n armazena o endereco de [x] = %p\n", &x);
  printf("\n&o armazena o endereco de [y] = %p\n", &y);
  printf("\n&z armazena o endereco de [z] = %p\n\n", &z);
        // &z retorna o endereco de onde o ponteiro z está
        // armazenado -> ainda um endereço -> %p é o especificador correto
  return 0;
}
```

Fundamentalmente, esse programa serve apenas para mostrar e exemplificar o uso e formato de declaração básicos de ponteiros.

2.2 Questão 2

inputencoding

```
    int ex2() {
  srand(time(NULL));
  int *ab, m = rand() % 200;

  printf("\nExercício 2 - Ponteiros: apresentação de interações entre "
         "ponteiros a "
         "partir\nda "
         "redefinição de valores:\n");
  printf("-----------------------------------------------------------------");
  printf("\n\nTemos a seguinte declaração:\nab = ponteiro int\ne digamos"
        "também que\nint m = %d\n\n", m);

  printf("Endereço de m:  [ %p ]\n", &m);
  printf("Valor de m:     ( %d )\n\n", m);
  printf("Valor de ab:    ( %d )\n", *ab);
  printf("Endereço de ab: [ %p ]\n\n", &ab);
  ab = &m;
  printf("Agora, ab (sem *) carrega o valor do endereço de m.\n\n");
  printf("Endereço do ponteiro ab (sem *):          [ %p ]\n", ab);
  printf("Conteúdo carregado pelo ponteiro ab (com *): %d\n\n", *ab);
  m = rand() % 100;
  printf("Agora, redefinimos o valor da variável m."
        "O novo valor designado agora é ( %d ).\n", m);
  printf("Endereço de ab (aponta para o endereço de m, como "
        "designado antes): [ %p ]\n", ab);
  printf("Conteúdo de ab: ( %d )\n\n", *ab);
  *ab = rand() % 50;
  printf("Agora, redefinimos que o valor da variável"
        "ab agora seja de %d.\n", *ab);
  printf("Endereço de m:  [ %p ]\n", &m);
  printf("Endereço de ab: [ %p ]\n", &ab);
  printf("\n\nObservação: Note que parenteses () foram utilizados para"
        "destacar números inteiros, enquanto colchetes foram usados para"
        "se referir a endereços de variáveis. Além disso, percebe-se"
        "que apesar de seus valores tenham sidos redesignados "
        "algumas vezes, observe que o endereço das"
        "variáveis m e ab sempre foram os mesmos.\n\n");
  printf("Valor final de m: ( %d )\n\n", m);
  return 0;
}
```

O objetivo deste exercício foi apresentar a manipulação básica de ponteiros e suas relações com as variáveis as quais eles se encontram associados, bem como a sua ligação com o endereço de memória alocado para aquela determinada variável.

2.3 Questão 3

inputencoding

```
    int ex3() {
  srand(time(NULL));
  int intX = rand() % 300;
  float floatX = 300.60;
  char cht = 65 + (random() % 27);
  // 65 = A, de acordo com a table ASCII. A adição é o fator responsável por
  // tornar a variável char aleatória

  printf("\n\nExercício 3 - Demonstração do uso dos operadores & e *\n");
  printf("-------------------------------------------------------\n\n");
  int *pt1;
  float *pt2;
  char *pt3;
  pt1 = &intX, pt2 = &floatX, pt3 = &cht;

  printf("intX   = %d\n", intX);
  printf("floatX = %f\n", floatX);
  printf("cht    = %c\n", cht);
  printf("\nUtilizando o operador &:\n");
  printf("-------------------------\n\n");
  printf("Endereço de intX   = %p\n", &intX);
  printf("Endereço de floatX = %p\n", &floatX);
  printf("Endereço de cht    = %p\n", &cht);
  printf("\nUtilizando os operadores & e *:\n");
  printf("-------------------------------\n\n");
  printf("Valor no endereço de intX   = %d\n", *(&intX));
  printf("Valor no endereço de floatX = %f\n", *(&floatX));
  printf("Valor no endereço de cht    = %c\n", *(&cht));
  printf("\nUtilizando apenas a variável ponteiro (%%p):\n");
  printf("---------------------------------------------\n\n");
  printf("Endereço de intX   = %p\n", pt1);
  printf("Endereço de floatX = %p\n", pt2);
  printf("Endereço de cht    = %p\n", pt3);
  printf("\nUtilizando apenas o operador ponteiro (*):\n");
  printf("-------------------------------------\n\n");
  printf("Valor no endereço de intX   = %d\n", *pt1);
  printf("Valor no endereço de floatX = %f\n", *pt2);
  printf("Valor no endereço de cht    = %c\n\n", *pt3);
  return 0;
}
```

Esse programa aprofunda no vínculo entre variável e endereço, por meio da declaração de variáveis do tipo ponteiros que "carregam"o endereço dos números aos quais elas também se encontram associados.

2.4 Questão 4

```
inputencoding
    int ex4() {
  int *nums = malloc(2 * sizeof(int));

  printf("\nExercício 4 - Adiçao de dois números utilizando ponteiros:\n");
  printf("-------------------------------------\n");

  printf("Insira o primeiro número: ");
  scanf("%d", nums);
  printf("Insira o segundo número: ");
  scanf("%d", (nums + sizeof(int)));

  printf("A soma dos números inseridos é: %d\n\n",
         *nums + *(nums + sizeof(int)));

  return 0;
}
```

Esse programa apresenta a adição de dois números através da utilização da função de alocação de memória, adicionado à lógica dos ponteiros. Note que apesar de mais de um número estar passando pela operação de soma, apenas uma variável é declarada no começo do código. Isso é possível por conta da alocação de mais de um espaço na memória do tipo de variável **int** para apenas uma variável, nesse caso representada por **nums**.

2.5 Questão 5

inputencoding

```
    long ex5_addNums(long *n1, long *n2) {
  long sum = *n1 + *n2;
  return sum;
}

int ex5() {
  long num1, num2;

  printf("\n\nExercício 5 - Adição de dois números a partir"
         "da chamada de variáveis "
         "(função de adição se encontra modularizada)\n");
  printf("-----------------------------------------------------------------");

  printf("Insira o primeiro número: ");
  scanf("%ld", &num1);
  printf("Insira o segundo número: ");
  scanf("%ld", &num2);
  printf("A soma de %ld e %ld é %ld\n\n", num1, num2,
         ex5_addNums(&num1, &num2));
  return 0;
}
```

Esse algoritmo, apesar de possuir a mesma funcionalidade que na questão anterior, utiliza um outro método para realizar a soma entre dois números. Esse método se trata da designação de valores às variáveis declaradas na função **ex5** e realiza as operações necessárias no módulo **addNums**.

2.6 Questão 6

```
inputencoding
int ex6_achaMaior(int *ptr1, int *ptr2) {

  if (*ptr1 > *ptr2) {
    printf("\n%d é o número maior.\n\n", *ptr1);
  } else {
    printf("\n%d é o número maior.\n\n", *ptr2);
  }
  return 0;
}

int ex6() {
  int fno, sno, *ptr1 = &fno, *ptr2 = &sno;

  printf("\n\nExercício 6 - Encontre o maior valor entre dois números:\n");
  printf("-----------------------------------------------------------------\n");

  printf("Insira o primeiro número: ");
  scanf("%d", ptr1);
  printf("Insira o primeiro número: ");
  scanf("%d", ptr2);
  ex6_achaMaior(&fno, &sno);
  return 0;
}
```

O objetivo desse programa foi de realizar uma comparação para avaliar se um número inserido é maior que o outro, por meio do módulo **achaMaior**.

2.7 Questão 7

inputencoding

```
    int ex7() {
  int arr1[25], i, n;
  printf("\n\nExercício 7 - Armazenar e fazer a chamada de"
         "elementos contidos em um vetor: \n");
  printf("-------------------------------------------------------------------------\n");
  printf("Insira o número de elementos que ãsero armazenados no vetor: ");
  scanf("%d", &n);

  printf("Insira %d elementos ao vetor:\n", n);
  for (i = 0; i < n; i++) {
    printf("Elemento - [ %d ]: ", i);
    scanf("%d", arr1 + i);
  }
  printf("Os elementos designados ao vetor ãso:\n");
  for (i = 0; i < n; i++) {
    printf("Elemento - [ %d ]: %d \n", i, *(arr1 + i));
  }
  return 0;
}
```

O algoritmo apresentado acima realiza o armazenamento de números dentro de um vetor e os imprime à tela, assim que o vetor é completamente preenchido.

2.8 Questão 8

inputencoding

```
    void ex8_changePosition(char *ch1, char *ch2) {
  char tmp;
  tmp = *ch1;
  *ch1 = *ch2;
  *ch2 = tmp;
}

void ex8_charPermu(char *cht, int stno, int endno) {
  int i;
  if (stno == endno)
    printf("%s  ", cht);
  else {
    for (i = stno; i <= endno; i++) {
      ex8_changePosition((cht + stno), (cht + i));
      ex8_charPermu(cht, stno + 1, endno);
      ex8_changePosition((cht + stno), (cht + i));
    }
  }
}

void ex8_randString(char *str, int num) {
  srand(time(NULL));
  int rando;
  for (int i = 0; i < num; i++) {
    // str[i] = rand() % ('z' - 'a' + 1) + 'a';
    int j = i + 1;
    str[i] = 97 + (random() % 26);
  }
  str[num] = 0;
}

int ex8() {
  char str[16];
  int c;
  printf("\n\nExercício 8 - Geração de permutações de uma dada string:\n");
  printf("-----------------------------------------------------------\n");
  printf("Quantas letras aleatórias êvoc deseja inserir à string? ");
  scanf("%d", &c);
  ex8_randString(str, c);
  int n = strlen(str);
  printf("As permutações da string ãso: \n");
  ex8_charPermu(str, 0, n - 1);
  printf("\n\n");
  return 0;
}
```

O código acima, apesar de ser longo, realiza a permutação entre letras (**char's**) apresentados em formato de **string**. Os processamentos são feitos por meio dos módulos acima da função principal (**ex8**) e os caracteres são gerados aleatoriamente, tendo seu alcance de 'a' até 'z'.

2.9 Questão 9

inputencoding

```
    int ex9() {
  int i, n;
  float *element;
  printf("\n\nExercício 9 - Encontre o maior elemento através"
         "da utilização de Memória âDinmica\n");
  printf("-----------------------------------------------------------------\n");
  printf("Selecione o número total de elementos: ");
  scanf("%d", &n);
  element =
      (float *)calloc(n, sizeof(float)); //A memória é alocada para 'n'
                                         //elementos inseridos pelo usuário
  if (element == NULL) {
    printf("ãNo há memória alocada o suficiente.\n");
    exit(0);
  }
  printf("\n");
  for (i = 0; i < n; ++i) {
    printf("Número %d: ", i + 1);
    scanf("%f", element + i);
  }
  for (i = 1; i < n; ++i) {
    if (*element < *(element + i))
      *element = *(element + i);
  }
  printf("O maior elemento é: %.2f \n\n", *element);
  return 0;
}
```

Similarmente ao exercício anterior, esse algoritmo também gera uma lista e designa seus valores por meio da interação do usuário. O programa, por sua vez, também realiza a organização da lista por meio do método de Locação de Memória Dinâmica.

2.10 Questão 10

inputencoding

```
int calculaTamanhoString(char* ch){
    int ctr = 0;
    while (*ch != ' ' && *ch != '\0'){
        ctr++;
        ch++;
    }
   if(*ch=='\0'){
     ctr--;
   }
    return ctr;
}
     int ex10() {
    char str1[25];
    int l;
          printf("Ponteiros: Calculando o tamanho de cada"
             "palavra de uma string:\n");
          printf("----------------------------------------\n");

    printf("Insira uma string: ");
    fgets(str1, sizeof str1, stdin);
    int i = 0;
    int wordIndex = 0;
    do{
        l=calculaTamanhoString(str1+i);
        i+=l+1;
        if(l!=0){
          wordIndex++;
          printf("\nTamanho da palavra %d: %d", wordIndex, l);
        }
    } while(*(str1+i)!='\0');

   return 0;
}
```

O código acima introduz uma maneira de calcular o tamanho de uma **string** por meio da utilização de chamada de de um módulo (calculaTamanhoString).

2.11 Questão 11

inputencoding

```
    void trocaNums(int *x,int *y,int *z) {
    int tmp;
    tmp=*y;
    *y=*x;
    *x=*z;
    *z=tmp;
}

int ex11()
{
    int e1,e2,e3;
        printf("\n10 - Trocar elementos por meio da chamada de função:\n");
        printf("-----------------------------------------------------\n");
    printf("Insira o valor do primeiro elemento:");
    scanf("%d",&e1);
        printf("Insira o valor do segundo elemento:");
    scanf("%d",&e2);
        printf("Insira o valor do terceiro elemento:");
    scanf("%d",&e3);


    printf("\nOs valores apresentados antes da troca ãso:\n");
    printf("Elemento 1 = %d\n Elemento 2 = %d\n"
           "Elemento 3 = %d\n",e1, e2, e3);
    trocaNums(&e1,&e2,&e3);
    printf("\nOs valores apresentados após a troca ãso:\n");
    printf(" Elemento 1 = %d\n Elemento 2 = %d\n"
           "Elemento 3 = %d\n\n", e1, e2, e3);
    return 0;
}
```

O objetivo do programa acima é de trocar os valores designados às variáveis cujos valores são inicialmente fornecidos pelo usuário no terminal por meio da função **scanf**. Isso é feito por meio da chamada da função trocaNums, que realiza a troca por meio da alteração do valor de cada variável uma pela outra.

2.12 Questão 12

inputencoding

```
    int ex12() {
  int fact;
  int num1;
  printf("\n\n12 - Ponteiros: Encontre o fatorial de um dado número,"
         "através da utilização de ponteiros:\n");
  printf("-------------------------------------------------------------\n");
  printf("Insira um número: ");
  scanf("%d", &num1);

  achaFat(num1, &fact);
  printf("O Fatorial de %d é: %d \n\n", num1, fact);
  return 0;
}
```

Nesse códigos utilizamos ponteiros como uma forma de acumular o valor dos cálculos feitos por meio do "for"utilizado do módulo acima.

2.13 Questão 13

inputencoding

```
    int ex13() {
  char str1[50];
  char *pt;
  int ctrV, ctrC;
  printf("\n\n13 - O ponteiro conta o número de vogais"
         "e consoantes da string: \n");
  printf("-------------------------------------------------\n");
  printf("Digite a string: ");
  fgets(str1, sizeof str1, stdin);

  pt = str1;
  // 0 para começar a busca no primeiro array.
  ctrV = ctrC = 0;

  while (*pt != '\0') {
   if (*pt == 'A' || *pt == 'E' || *pt == 'I' || *pt == 'O' || *pt == 'U' ||
       *pt == 'a' || *pt == 'e' || *pt == 'i' || *pt == 'o' || *pt == 'u')
     ctrV++;
   else
     ctrC++;
   pt++; // O número do ponteiro aumenta de acordo cada váriavel para
         //  buscar por consoantes/vogais.
  }

  printf(" Número de vogais : %d\n Número de consoantes : %d\n", ctrV,
         ctrC - 1);
  return 0;
}
```

Nesse código usamos ponteiros como uma **array** que, por meio da estrutura de repetição **while** utilizada no código, que aumenta o valor de uma variável de consoantes e vogais caso uma determinada ação específica ocorrer.

2.14 Questão 14

```
inputencoding
    int ex14() {
  int *a, i, j, tmp, n;
  printf("\n\n14 - Rearranjando Array com Ponteiros :\n");
  printf("-----------------------------------------------\n");

  printf(" Qual o tamanho do array: ");
  scanf("%d", &n);

  printf(" Qual o número do elemento %d da array : \n", n);
  for (i = 0; i < n; i++) {
    printf(" elemento - %d : ", i + 1);
    scanf("%d", a + i);
  }
  for (i = 0; i < n; i++) {
    for (j = i + 1; j < n; j++) {
      if (*(a + i) > *(a + j)) {
        tmp = *(a + i);
        *(a + i) = *(a + j);
        *(a + j) = tmp;
      }
    }
  }
  printf("\n Os elementos das arrays agora ãso : \n");
  for (i = 0; i < n; i++) {
    printf(" elemento - %d : %d \n", i + 1, *(a + i));
  }
  printf("\n");
  return 0;
}
```

Nesse código utilizamos Ponteiros para rearranjar quais são os valores da **array**, fazemos isso por meio da estrutura de condição **if** que usamos para ver se algumas variáveis são maiores que outras, e com isso alterar o valor do ponteiro e portanto, do valor que está dentro de uma **array**.

2.15 Questão 15

inputencoding

```
    int *ex15_findLarger(int *n1, int *n2) {
  if (*n1 > *n2)
    return n1;
  else
    return n2;
}

int ex15() {
  int *ex15_findLarger(int *, int *);
  int numa = 0;
  int numb = 0;
  int *result;
  printf("\n\n 15 - Mostre uma função usando ponteiros :\n");
  printf("--------------------------------------------------\n");
  printf(" Escolha o primeiro número : ");
  scanf("%d", &numa);
  printf(" Escolha o segundo número : ");
  scanf("%d", &numb);
  result = ex15_findLarger(&numa, &numb);
  printf(" O número %d é maior.  \n\n", *result);
  return 0;
}
```

Nesse código usamos ponteiros para detectar se um número é maior que o outro a partir de estruturas de condição **if** que usamos, e com o ponteiro apontando para uma variável que significa o resultado delas.

2.16 Questão 16

inputencoding

```
    int ex16() {
  int arr1[10];
  int i, n, sum = 0;
  int *pt;

  printf("\n\n16 - Somando todos os elementos de uma"
         "array com ponteiro:\n");
  printf("-----------------------------------------------------\n");

  printf(" Qual o tamanho da array (máximo 10) ");
  scanf("%d", &n);

  printf(" Selecione os %d números da array: \n", n);
  for (i = 0; i < n; i++) {
    printf(" elemento - %d : ", i + 1);
    scanf("%d", &arr1[i]);
  }

  pt = arr1; // pt guarda o endereço de arr1

  for (i = 0; i < n; i++) {
    sum = sum + *pt;
    pt++;
  }

  printf(" A soma do array é : %d\n\n", sum);
  return 0;
}
```

Nesse código utilizamos de ponteiros para somar a soma do valor guardado em cada casa da **array**, com o ponteiro apontando inicialmente para a primeira casa e, a partir da estrutura de repetição **for**, começa a apontar para cada casa por vez, até somar todos os valores.

2.17 Questão 17

inputencoding

```
    int ex17() {
  int n, i, arr1[15];
  int *pt;
  printf("\n\n17 - Escrevendo os números da array em ordem inversa:\n");
  printf("-----------------------------------------------------------------\n");

  printf(" Qual o tamanho da array (máximo 15) ");
  scanf("%d", &n);
  pt = &arr1[0]; // pt guarda o endereço de arr1
  printf(" Escreva os %d números da array: \n", n);
  for (i = 0; i < n; i++) {
    printf(" Elemento - %d : ", i + 1);
    scanf("%d", pt); // Endereço do valor
    pt++;
  }

  pt = &arr1[n - 1];

  printf("\n Os elementos da array em ordem reversa é:");

  // Escrevendo os Valores ja definidos com o for ao contrário (do final da
  // array para baixo)

  for (i = n; i > 0; i--) {
    printf("\n element - %d : %d  ", i, *pt);
    pt--;
  }
  printf("\n\n");
  return 0;
}
```

Nesse código utilizamos de ponteiros para imprimir a **array** de forma inversa, do final até o início, com o ponteiro apontando inicialmente para a primeira casa até a ultima a partir da estrutura de repetição **for** para definir os valores, porém depois usando **printf** dentro de outro **for** para fazer o oposto, com o ponteiro apontando para a casa final do **array** até a primeira.

2.18 Questão 18

inputencoding

```
    struct ex18_EmpAddress {
  char *ename;
  char stname[20];
  int pincode;
} employee = {"John Alter", "Court Street \n", 654134}, *pt = &employee;

int ex18() {
  printf("\n\n 18 - Uso de ponteiros em estruturas :\n");
  printf("-----------------------------------------------------------\n");
  printf(" %s de %s \n\n", pt->ename, (*pt).stname);
  return 0;
}
```

Nesse código o ponteiro aponta para uma variável que guarda outras variáveis, esse conceito se chama **structure**, que então são escolhidas a partir desse ponteiro para descrever o que está pedindo.

2.19 Questão 19

inputencoding

```
    union empAdd
{
char *ename;
char stname[20];
int pincode;
};

int ex19()
{
        printf("\n\n Mostrando ponteiro com union:\n");
        printf("-----------------------------------------\n");
    union empAdd employee,*pt;
    employee.ename="Jhon Mc\0Donald";
    //Unir char com caracteres null '\0'

    pt=&employee;

    printf(" %s %s\n\n",pt->ename,(*pt).ename);

    return 0;
}
```

Nesse código vemos algo parecido com o ultimo exercício, porém agora com o conceito de união em vez de estruturas.

2.20 Questão 20

inputencoding

```
    struct employee
{
char *empname;
int empid;
};

int ex20()
{
     printf("\n\nUsando Ponteiros que apontam para uma"
            "array que guarda uma structure:\n");
    printf("-------------------------------------------------");

    static struct employee emp1 =
    {"Jhon",1001},emp2={"Alex",1002},emp3={"Taylor",1003};
    struct employee(*arr[])={&emp1, &emp2, &emp3};
    struct employee(*(*pt)[3])=&arr;

    printf(" Nome do Empresário : %s \n",(**(*pt+1)).empname);
    printf("------------------ Explicação ------------------\n");
    printf("(**(*pt+1)).empname\n");
    printf("= (**(*&arr+1)).empname    as pt=&arr\n");
    printf("= (**(arr+1)).empname      de acordo com *&pt = pt\n");
    printf("= (*arr[1]).empname        de acordo com *(pt+i) = pt[i]\n");
    printf("= (*&emp2).empname         as arr[1] = &emp2\n");
    printf("= emp2.empname = Alex        de acordo com *&pt = pt\n\n");
    printf(" Employee ID :  %d\n",(*(*pt+1))->empid);
    printf("------------------ Explicação ------------------\n");
    printf("(*(*pt+1))-> empid\n");
    printf("= (**(*pt+1)).empid      de acordo com -> = (*).\n");
    printf("= emp2.empid = 1002\n");
    printf("\n\n");
    return 0;
}
```

Esse código se utiliza de ponteiros para apontar em uma **array** que guarda estruturas, juntando então as explicações de **array** e de uniões com ponteiros, com o ponteiro apontando para uma **array**, que guarda uma estrutura, que guarda varáveis que então são utilizadas no código.

2.21 Questão 21

inputencoding

```
    int ex21() {
  char alph[27];
  int x;
  char *ptr;
  printf("\n\n 21 - Escrevendo o alfabeto com ponteiros:\n");
  printf("-----------------------------------------\n");
  ptr = alph;
  // Alocando as letras do Alfabeto para seus
  // respectivos endereços na array
  for (x = 0; x < 26; x++) {
    *ptr = x + 'A';
    ptr++;
  }
  ptr = alph;
  // Escrevendo o Alfabeto
  printf(" O alfabeto é : \n");
  for (x = 0; x < 26; x++) {
    printf(" %c ", *ptr);
    ptr++;
  }
  printf("\n\n");
  return (0);
}
```

Nesse exercício utilizamos mais uma vez o ponteiro como uma forma de **array**, no qual definimos primeiro seu valores (com o ponteiro) com a função de repetição **for** para que cada função de ponteiro tenha um valor e então utilizamos a mesma função de repetição **for** para dessa vez usar **printf** neles.

2.22 Questão 22

inputencoding

```
    int ex22() {

  char str1[50];
  char revstr[50];
  char *stptr = str1;
  char *rvptr = revstr;
  int i = -1;
  printf("\n\n 22 - String ao invertida com ponteiros :\n");
  printf("-----------------------------------------------------\n");
  // Definindo a String
  printf(" Escreva uma palavra : ");
  scanf("%s", str1);
  while (*stptr) {
    stptr++;
    i++;
  }
  // Invertendo ela
  while (i >= 0) {
    stptr--;
    *rvptr = *stptr;
    rvptr++;
    --i;
  }
  *rvptr = '\0';
  printf(" A string reversa é : %s\n\n", revstr);
  return 0;
}
```

Nesse código, mais uma vez, utilizaremos do ponteiro como uma **array**, ou nesse caso melhor dizendo uma **string**, para que cada letra ou caracteres seja o valor de uma das casas, e então usamos de outro ponteiro, que aponta para o ponteiro inicial, para escrever a **array** do final até o início, conseguindo então escrever a **array** de trás para frente.

3 CONCLUSÃO

A partir do aprofundamento nesse trabalho de investigação de Ponteiros, fomos capazes de adquirir uma compreensão melhor, mais ampla e clara da aplicação, do uso e da prática de pointers, em C. Fora isso, também adquirimos conhecimento a respeito da elaboração de um arquivo de apresentação e defesa dos códigos aqui apresentados.

Referências: PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS, Padrão PUC Minas de Normalização, Disponível em: <http://www.pucminas.br/biblioteca/>. Acesso em: 6 de Set. 2013.

W3Resources - Pointer :
https://www.w3resource.com/c-programming-exercises/pointer/index.php

REFERÊNCIAS