

Ponteiros

1. **Memória.** Fazer um esquema representando a memória do computador ao se executar a sequência de comandos.

```
int var;  
int *ptr;  
var = 10;  
ptr = &var;
```

2. **Ponteiro.**

Um ponteiro é:

- (a) O endereço de uma variável.
 - (b) Uma variável que armazena endereços de memória.
 - (c) O valor de uma variável.
 - (d) Um indicador da próxima variável a ser acessada.
3. **Endereço.** Escreva uma instrução que imprima o endereço da variável `var`.
 4. **Operador*.** O que significa o operador `*` (asterisco) em cada um dos seguintes casos:
 - (a) `int *p;`
 - (b) `cout << *p;`
 - (c) `*p = x*5;`
 - (d) `cout << *(p+1);`
 5. **Declaração.** Quais das seguintes instruções declaram um ponteiro para uma variável **float**:
 - (a) `float *p;`
 - (b) `*float p;`
 - (c) `float* p;`
 - (d) `float *p=&f;`
 - (e) `*p;`
 6. **Expressões.** Se o endereço de **var** foi atribuído a um ponteiro **pvar**, quais das seguintes expressões são verdadeiras?
 - (a) `var == &pvar`
 - (b) `var == *pvar`
 - (c) `pvar == *var`
 - (d) `pvar == &var`
 7. **Declarações.** Considere as declarações:

```
int i = 3, j = 5  
int *p = &i, *q = &j;
```

Indique qual é o valor das seguintes expressões:

- (a) `p == &i`
- (b) `*p - *q`
- (c) `**&p`
- (d) `3*-p/(q)+7`

8. Qual a saída do programa?.

```
#include <iostream>

void main(){
    int i = 5, *p;
    p = &i;
    cout<<p<<"\t"<<(*p+2) <<"\t"<< **&p<<"\t"<< (3**p) <<"\t"<< (**&p + 4);
    return 0;
}
```

9. **Atribuições.** Se **i** e **j** são variáveis inteiras e **p** e **q** ponteiros para **int**, quais das seguintes expressões de atribuição são incorretas?

- (a) `p == &i`
- (b) `*q == &j`
- (c) `p == *&i`
- (d) `i = (*&j`
- (e) `i = *&*&j`
- (f) `q = &p`
- (g) `i = (*p)++ + *q`
- (h) `if (p==i) i++`

10. **Erro.** O seguinte programa está correto? Justifique.

```
#include <iostream>
#define VAL 987

int main(){
    int *p = VAL;
    cout<<*p
    return 0;
}
```

11. **Vetores e ponteiros.**

- (a) Qual a diferença entre `vet[3]` e `*(vet +3)`?
- (b) Admitindo a declaração: `int vet[8];`, por que a instrução `vet++` é incorreta?
- (c) Admitindo a declaração `int vet[8];`, quais das seguintes expressões referenciam o valor do terceiro elemento do vetor?

1. *(vet+2)
2. *(vet+3)
3. vet+2
4. vet+3

12. **Saída.** Qual a saída de cada um dos seguintes programas?

```
#include <iostream>

int main(){
    int mat[] = {4, 5, 6};
    for(int i=0; i<3; i++)
        cout<<*(mat + i)<<endl;
    return 0;
}
```

```
#include <iostream>

int main(){
    int mat[] = {4, 5, 6};
    for(int i=0; i<3; i++)
        cout<<mat + i<<endl;
    return 0;
}
```

```
#include <iostream>

int main(){
    int mat[] = {4, 5, 6};
    int *p=mat;
    for(int i=0; i<3; i++)
        cout<<*p++<<endl;
    return 0;
}
```

13. Qual a diferença entre as duas instruções seguintes?

```
char s[] = "Brasil";
char *s = "Brasil";
```

14. **Saída.** Assumindo a declaração:

```
char *s = "Eu nao vou sepultar cesar";
```

O que será impresso em cada uma das instruções:

(a) cout<<s

- (b) `cout<<(void*)&s[0];`
 - (c) `cout<<(void*)(s+11);`
 - (d) `cout<<s[0];`
 - (e) `cout<<(void*)s`
15. **Aritmética de ponteiro.** Escreva a expressão `mat[i][j]` usando notação de ponteiro (aritmética de ponteiro).
16. **Funções.** Qual é a diferença entre os seguintes protótipos de funções:

```
void func (char *p);  
void func (char []);
```

17. **Declaração.** Assumindo a declaração:

```
char *items[5] = {"Abrir", "Fechar", "Salvar", "Imprimir", "Sair"};
```

Para poder escrever a instrução `p = items;`, a variável `p` deve ser declarada como:

- (a) `char p;`
- (b) `char *p;`
- (c) `char **p;`
- (d) `char ***p;`

Alocação Dinâmica

18. **Vetor de tamanho n.** Faça um programa (função `main`) que leia um número inteiro `n` e que aloque dinamicamente usando o operador `new[]` da linguagem C++ um vetor de números reais usando o tipo de dados `float`. Faça um loop para ler cada um dos valores do vetor e em seguida calcule a média dos valores e armazene-a na variável `media`. Por fim, libere a memória alocada dinamicamente usando o operador `delete[]`.
19. **Função calculaMedia.** Estenda o programa do exercício anterior criando agora uma função chamada `calculaMedia` que recebe como parâmetro um número inteiro `n` e um ponteiro para `float`, representando o número de elementos do vetor e o endereço do seu primeiro elemento, respectivamente. A função deve calcular a média dos valores do vetor e retornar esse valor. A função tem o seguinte protótipo:

```
float calculaMedia(int n, float *vet);
```

20. **Função para alocar vetores.** Crie sua própria função para alocar dinamicamente um vetor de números reais usando o tipo de dados `float` e `double`. As funções devem obedecer aos seguintes protótipos:

```
float* alocaVetorF(int n);  
double* alocaVetorD(int n);
```

Onde **n** indica o número de elementos a ser alocado. Sua implementação deve usar internamente o operador **new[]** para fazer a alocação. O operador **new[]** retorna o endereço de memória do primeiro elemento ou caso aconteça algum problema esta retorna o valor **NULL**. Sendo assim, verifique na sua função se não houve nenhum problema e retorne o ponteiro para o primeiro elemento, caso contrário imprima uma mensagem dizendo que não foi possível alocar a memória solicitada. **Obs:** as funções criadas poderão ser usadas da seguinte forma:

```
float *vf;  
double *vd;  
vf = alocaVetorF(100);  
vd = alocaVetorD(1000);
```

21. **Procedimento para alocar vetores.** Crie agora um **procedimento** para alocar vetores de tamanho **n**, de forma similar ao exercício anterior. Entretanto, agora use o seguinte protótipo:

```
void alocaVetorF(int n, float *v);  
void alocaVetorD(int n, double *v);
```

Obs: os procedimentos criados poderão ser usados da seguinte forma:

```
float *vf;  
double *vd;  
alocaVetorF2(100, vf);  
alocaVetorD2(100, vd);
```

22. **Soma de vetores.** Crie uma função que receba dois ponteiros (vetores) do tipo **float** e um número inteiro **n** indicando o tamanho dos vetores. Esta função deve:

- Verificar se os dois vetores tem o mesmo tamanho **n**.
- Alocar um novo vetor **result** de tamanho **n** de forma dinâmica.
- Calcular a soma dos vetores e armazenar em **result**.
- Retornar o vetor (i.e. ponteiro para o primeiro elemento do vetor) que contem os resultados.

A função possui o seguinte protótipo:

```
float* calculaSoma(int n, float a[], float b[]);
```

23. **Matriz com vetor de ponteiros.** Crie uma função que aloque dinamicamente uma matriz de tamanho **m x n** de valores do tipo **float** e inicialize todos os seus elementos com o valor 0. Essa função deve obedecer ao seguinte protótipo:

```
float** alocaMatrizF(int m, int n);
```

24. **Matriz transposta.** Implemente uma função que receba uma matriz **m x n** de valores do tipo **float**, crie e retorne sua matriz transposta **n x m**. A função tem o seguinte protótipo:

```
float** transposta(int m, int n, float **mat);
```

Recursividade

25. Escreva uma função recursiva que calcule $1 + 2 + 3 + \dots + n$, isto é, a soma dos números de 1 até n , onde n é dado como parâmetro para a função.

```
int soma(int n);
```

26. Escreva uma função recursiva que calcule a soma dos números de a até b , onde a e b são dados como parâmetro para a função e são tais que $a < b$.

```
int soma(int a, int b);
```

27. O número de dígitos de um número inteiro positivo pode ser determinado através de sucessivas divisões por 10 (sem guardar o resto) até que o número seja menor do que 10, consistindo de apenas 1 dígito. Implemente uma função recursiva que calcule o número de dígitos de um inteiro positivo n .

```
int numDigitos(int n);
```

28. Escreva uma função recursiva que determina se um vetor de caracteres é um palíndromo. A função recebe como parâmetros o tamanho e o vetor e deve retornar **true** ou **false**.

```
bool ehPalindromo(char a[], int n);
```

29. Escreva uma função recursiva que faça a busca por uma chave (um valor específico) em um array ordenada usando o algoritmo da busca binária. A função recebe como parâmetros o array de inteiros, seu tamanho e a chave a ser procurada e deve retornar se encontrou ou não o valor da chave.

```
bool buscaBinaria(int vet[], int n, int chave);
```

Complexidade e notação assintótica

30. Determinar a complexidade para cada um dos seguintes fragmentos de programa em C/C++.

(a)

```
for(int i=0; i<n; ++i)
    ++k;
```

(b)

```
for(int i=1; i<n; i*=2)
    ++k;
```

(c)

```
for(int i=n-1; i!=0; i/=2)
    ++k;
```

(d)

```
for(int i=0; i<n; ++i)
    if(i%2 == 0)
        ++k;
```

(e)

```
for(int i=0; i<n; ++i)
    for(int j=0; j<n; ++j)
        ++k;
```

(f)

```
for(int i=0; i<n; ++i)
    for(int j=i; j<n; ++j)
        ++k;
```

(g)

```
for(int i=0; i<n; ++i)
    for(int j=0; j<i*i; ++j)
        ++k;
```

31. Construir uma função recursiva para calcular o fatorial de n de acordo com a equação:

$$n! = \begin{cases} 1, & n = 0, \\ n \times (n-1)!, & n > 0. \end{cases}$$

Calcular o tempo de processamento da função.

32. Considere o trecho de programa abaixo. Que valor é calculado pela função f ? (Dê sua resposta em função de n .) Dê uma expressão justa de O para o tempo de processamento no pior caso da função f . Faça uma função $f1$ que seja $O(1)$.

```
int f(int n)
{
    int soma = 0;
    for(int i=1; i<=n; ++i)
        soma = soma + 1;
    return soma;
}
```

33. Considere o trecho de programa abaixo. (a função f é dada no exercício acima.) Que valor é calculado por g ? (Dê sua resposta em função de n .) Dê uma expressão justa de O para o tempo de processamento no pior caso do método g .

```
int g(int n)
{
    int soma = 0;
    for(int i=1; i<=n; ++i)
        soma = soma + i + f(i);
    return soma;
}
```

34. Considere o trecho de programa abaixo. (as funções f e g são dadas nos exercícios acima.) Que valor é calculado por h ? (Dê sua resposta em função de n .) Dê uma expressão justa de O para o tempo de processamento no pior caso do método h .

```
int h(int n)
{
    return f(n) + g(n);
}
```

35. Escreva um procedimento/função que receba um argumento inteiro n e tenha um tempo de processamento no pior caso:

- (a) $O(n)$;
- (b) $O(n^2)$;
- (c) $O(n^k)$. Onde k também é entrada do algoritmo;
- (d) $O(\log n)$
- (e) $O(n \log n)$
- (f) $O(2^n)$

Tipos Abstrados de Dados (TADs)

36. **Centro Geométrico.** Considerando o TAD Ponto implementado, crie uma função que receba um vetor de pontos (**Ponto *v**) e calcule o centro geométrico desse conjunto de pontos usando a seguinte expressão:

$$\bar{x} = \frac{\sum x_i}{n}, \bar{y} = \frac{\sum y_i}{n}$$

Onde x_i e y_i representam as coordenadas **x** e **y** do ponto **i** e, \bar{x} e \bar{y} as coordenadas do ponto que representa o centro geométrico. Essa função deve alocar dinamicamente o ponto que representa o centro geométrico e retornar o ponteiro para esse ponto ao final do processamento.

37. **TAD Círculo.**

Abaixo encontra-se parte do arquivo circulo.h que implementa a TAD Círculo. Os dados armazenados para representar um círculo são o seu raio e seu centro, sendo o centro do tipo Ponto (TAD Ponto já implementado):

Arquivo circulo.h

```
class Circulo {
private:
    float raio;
    Ponto *centro;
public:
    Circulo (float r, float x, float y);
};
```

Considerando o TAD Círculo acima, desenvolver em C++ o construtor o qual cria um círculo dados as coordenadas **x** e **y** e o raio **r**. Além disso desenvolver os métodos **area_circulo** que calcula e retorna a área de um círculo, e a função **interior_circulo** que verifica se **p** está no

interior de um círculo **c** e retorna **true** ou **false**. Considerar que as seguintes funções do TAD Ponto, estão implementadas:

```
Ponto(float x, float y); // construtor
float distanciaPonto(Ponto *p1, Ponto *p2);
```

Protótipos dos métodos a serem implementados:

```
float getArea(Circulo *c);
bool interiorCirculo(Ponto *p);
```

38. **TAD Quadrado.** Usando o TAD Ponto, implementado em sala de aula, implemente o TAD Quadrado.

- Criar um quadrado.
- Liberar a memória ocupada por quadrado.
- Acessar um ponto do quadrado.
- Atribuir/alterar um ponto do quadrado.
- Consultar se um ponto x está dentro ou fora do quadrado.

39. **TAD Triângulo.** Usando o TAD Ponto, implementado em sala de aula, implemente o TAD Triangulo usando 3 objetos do tipo Ponto para representar os vértices **v1**, **v2** e **v3** do triângulo. Implemente as operações:

- Criar um triângulo.
- Liberar a memória ocupada por um triângulo.
- Acessar um ponto do triângulo.
- Atribuir/alterar um ponto triângulo.
- Consultar se um ponto x está dentro ou fora do triângulo.

40. **TAD Racional.** Implemente um tipo abstrato de dados para representar números racionais. Um número racional é o quociente de dois números inteiros. A definição do tipo é apresentada a seguir:

```
class Racional {
    private:
        int numerador, denominador;
    public:
        Racional(int n, int d);
        Racional* soma(Racional *rac2);
        Racional* subtracao(Racional *rac2);
        Racional* multiplicacao(Racional *rac2);
        Racional* divisao(Racional *rac2);
        void imprime(Racional *rac);
        bool ehIgual(Racional *rac);
};
```

TADs - Domínios

41. Qual a diferença entre TAD e domínio de dado?
42. Formular definições por enumeração para os domínios:
 - (a) Estações do ano.
 - (b) Meses do ano.
43. Definir domínios apropriados para (usar `typedef` ou `class`):
 - Retas, dadas por dois de seus pontos;
 - Endereços, dados por rua e número;
 - Voos, dados por distância e tempo de duração;
 - Ponto no globo terrestre, dados por latitude e longitude;
 - Contas bancárias, das por número, saldo e nome do correntista.
 - Carros nacionais, com as informações: identificação (composta de estado e placa), ano de registro e se tem infrações registradas;
 - Carros estrangeiros, com as informações: país de origem, data de entrada e placa.
44. Definir os seguintes domínios (vetores)
 - Polinômio com coeficientes reais: $5.2x^3 + 4.7x^2 + 2.0x + 4.0$.
 - Uma matriz 3x5 de reais pode ser vista como um vetor composto de 3 linhas, cada uma delas sendo, por sua vez, um vetor com 5 componentes reais. Defina um domínio baseado nesta ideia.
45. Um polinômio nas variáveis x , y e z consiste de vários termos. Cada termo pode ser visto como uma quádrupla consistindo do expoente de x , y e z e do coeficiente do termo. Formular uma definição de domínio para representar um polinômio com 5 termos.

Programação com TAD's

46. Desenvolver o TAD Ponto 2D (Ponto2D) com coordenadas (X, Y) que deve ser capaz de realizar as seguintes operações:
 - (a) Criar (construtor) o TAD Ponto2D, dadas as coordenadas X e Y;
 - (b) Imprimir a coordenada X do ponto Ponto2D;
 - (c) Imprimir a coordenada Y do ponto Ponto2D;
 - (d) Calcular a distancia entre os pontos P e Q. A distância entre dois pontos P(X1,Y1) e Q(X2,Y2) é dada por: $d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$
47. Desenvolver o TAD Circulo, que deve ser representado pelo seu centro - que deve ser do tipo Ponto2D - e pelo raio (um número real). O TAD Circulo deve ser capaz de realizar as seguintes operações:
 - (a) Criar o TAD Circulo (construtor), dados o ponto representando o centro e o raio do círculo;

- (b) Imprimir as coordenadas do centro;
- (c) Imprimir o raio do centro;
- (d) Calcular a área do círculo;
- (e) Calcular o perímetro do círculo.

Desafio: desenvolver uma nova operação (outro construtor) para criar o TAD Circulo acima dados 3 pontos do tipo Ponto2D. Neste caso, deve-se calcular o ponto do centro e o raio do círculo e depois chamar o construtor desenvolvido em (a) - onde os parâmetros são o centro e o raio calculados.

48. Desenvolver o TAD (Triangulo), usando o TAD Ponto2D desenvolvido acima. O construtor do TAD deve avaliar se três pontos definem um triângulo antes de criar o TAD Triângulo. O TAD Triângulo deve ser capaz de realizar as seguintes operações:
 - (a) Criar o TAD (construtor) com os três pontos que representam os vértices (testando se eles formam um triângulo);
 - (b) Calcular e retornar a área do triângulo;
 - (c) Calcular e retornar o perímetro do triângulo;
 - (d) Retornar uma mensagem dizendo se o triângulo é equilátero, isósceles ou escaleno.
49. Desenvolver o TAD VetorDeTriangulos que representa um vetor de 50 posições contendo 50 TAD Triangulo. O TAD deve possuir um procedimento Adiciona(), que adiciona um triângulo na próxima posição vazia do vetor, se houver espaço, ou retorna uma mensagem de erro caso não haja.
50. Faça um programa (PA) em C++ em que usuário entre com o número de círculos e o número de triângulos (definidos anteriormente). O programa deve alocar dinamicamente um vetor para guardar a quantidade de círculos e de triângulos digitados pelo usuário. DICA: Lembre-se que um vetor é um ponteiro para o primeiro elemento deste vetor.
51. Construir o TAD Matriz2D para representar uma matriz bidimensional $m \times n$ e que realiza as seguintes operações:
 - (a) Criar a matriz dados m e n e iniciar com zero seus elementos;
 - (b) Consultar um elemento a partir do par de índices L e C;
 - (c) Atribuir um elemento na posição L e C da matriz;
 - (d) Imprimir na tela a matriz;
 - (e) Imprimir na tela a matriz transposta de uma matriz qualquer;
 - (f) Multiplicar uma matriz por outra e exibir na tela o resultado.
52. Baseado no exercício 40. Um monômio nas variáveis x, y e z consiste de 3 valores inteiros positivos representando os expoentes de x, y e z e de um valor real representando o coeficiente do monômio. Exemplos de monômios:

$$3x^2yz, xyz, 4x, 9.5, 4.0xy^5z, -5.2z^3y^5x^2$$

Desenvolver o TAD Monomio para representar um monômio como definido acima e que tenha as seguintes operações:

- (a) Construtores considerando as 5 diferentes variações:
1. nenhum parâmetro (colocar o valor 1.0 para o coeficiente e zero para os expoentes);
 2. apenas o coeficiente (colocar zero para todos os expoentes);
 3. o coeficiente e o expoente de x (colocar zero para os demais expoentes);
 4. o coeficiente e os expoentes de x e y (colocar zero para expoente z);
 5. o coeficiente e os expoentes de x, y e z;

Observe que não pode haver coeficiente 0.0 (zero).

- (b) Consultar o valor do coeficiente e o valor de qualquer um dos expoentes;
- (c) Atribuir um valor ao coeficiente (diferente de zero) e a qualquer um dos expoentes;
- (d) Avaliar um monômio considerando as diferentes possibilidades:
1. nenhum parâmetro (considerar o valor 1.0 para x, y e z);
 2. apenas o valor de x (considerar 1.0 para y e z);
 3. os valores de x e y (considerar 1.0 para z);
 4. os valores de x, y e z.
- (e) Verificar se dois monômios são iguais;
- (f) Calcular e retornar a derivada de um monômio. Usar índices 1, 2 e 3 para as variáveis x, y e z, respectivamente. Assim, seja m um monômio, `m->Derivada(2)` e `m->Derivada(3)` calcula e retorna outro monômio como sendo a derivada de y e z, respectivamente
- (g) Calcular e retornar a integral de um monômio. Usar índices 1, 2 e 3 para as variáveis x, y e z, respectivamente. Assim, seja m um monômio, `m->Integral(2)` e `m->Integral(3)` calcula e retorna outro monômio como sendo a integral de y e z, respectivamente.
- (h) Imprimir um monômio. Se um variável (x, y ou z) tem expoente zero não deve imprimi-la, se tem expoente 1 não imprimir tal expoente. Nos demais casos imprimir o sinal ^ como exponenciação. Exemplos: $3x^2yz$, xyz , $4x$, 9.5 , $4.0xy^5z$, $-5.2x^3y^5z^2$;
- (i) Calcular e retornar a multiplicação de dois monômios.

53. Baseado no exercício 47. Um polinômio nas variáveis x, y e z consiste de vários monômios, ver exercício 47. Exemplos de polinômios:

$$3x^2yz + xyz - 4.0x + 4.0xy^5z - 5.2z^3y^5x^2 + 9.5$$

Desenvolver o TAD `Poli_xyz` para representar um polinômio com no máximo 100 monômios (usar um vetor de tamanho 100, o polinômio será construído adicionando monômios) e que tenha as seguintes operações:

- (a) Construtores considerando as 3 diferentes possibilidades:
1. nenhum parâmetro; cria um polinômio vazio;
 2. um monômio; cria um polinômio com um monômio;
 3. o coeficiente e os expoentes de x, y e z; cria um polinômio com um monômio;
- (b) Adicionar um monômio ao polinômio. Cuidado para não adicionar monômios iguais. Sendo assim, verificar se há monômio igual ao que se deseja adicionar.
- (c) Consultar o valor do coeficiente que está no i-ésimo termo (um monômio) do polinômio;

- (d) Atribuir um valor ao coeficiente (diferente de zero) e a qualquer um dos expoentes que estão no i-ésimo termo (um monômio) do polinômio;
- (e) Avaliar um polinômio dados os valores de x, de y e de z.
- (f) Calcular e retornar a derivada de um polinômio. Usar índices 1, 2 e 3 para as variáveis x, y e z, respectivamente. Assim, seja p um polinômio, `p->Derivada(2)` e `p->Derivada(3)` calcula e retorna outro polinômio como sendo a derivada de em relação a y e a z, respectivamente.
- (g) Calcular e retornar a integral de um polinômio. Usar índices 1, 2 e 3 para as variáveis x, y e z, respectivamente. Assim, seja p um polinômio, `p->Integral(2)` e `p->Integral(3)` calcula e retorna outro polinômio como sendo a integral de y e z, respectivamente.
- (h) Imprimir um polinômio. Se um variável (x, y ou z) tem expoente zero não deve imprimi-la, se tem expoente 1 não imprimir tal expoente. Nos demais casos imprimir o sinal ^ para exponenciação. Exemplos: $3x^2yz + xyz - 9.5 + 4.0xy^5z - 5.2x^3y^5z^2$;
- (i) Calcular e retornar a soma de dois polinômios.
- (j) Calcular e retornar a subtração de dois polinômios.