Linguagem de Programação

Prof. Isaac Benchimol

Lista de Exercícios 2

1. Considere as seguintes declarações de variáveis:

int x = 10;

int var[] = {30, 10, 20, 40, -10, 0, 10, -20, 40, 30};

char str[20] = {"Minha string"};

int numero1 = 10, numero2 = 20;

1. Utilize um ponteiro para mostrar o valor de **x**
2. Mostre o endereço de **x**
3. Utilize um ponteiro para somar e mostrar a soma dos elementos de **var**
4. Utilize um ponteiro para mostrar a string **str**
5. Utilize um ponteiro para trocar os valores de **numero1** e **numero2**
6. Considere a seguinte classe:

class estudante {

protected:

int matricula, N1, N2;

public:

void **get**() {

cout << "Entre Matricula :";

cin>> matricula;

cout << "Entre notas :";

cin >> N1 >> N2;

}

};

1. Declare um ponteiro para **estudante** e utilize-o para atribuir valores aos atributos de um objeto
2. Crie uma função pública para **estudante** que mostre os seus atributos protegidos.
3. Use alocação dinâmica de memória para criar um objeto **estudante**. Atribua valores para os atributos deste objeto.
4. Use alocação dinâmica de memória para alocar memória para 100 valores reais 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, .... Mostre a soma dos valores.
5. Crie uma classe **Conta** com o atributo **saldo** e funções para mostrar o saldo, depositar e sacar. Inclua também uma função para atualizar o saldo da conta de acordo com uma taxa percentual fornecida. Crie duas sublasses de Conta: **ContaCorrente** e **ContaPopupança**. Reescreva a função (utilizando **polimorfismo**) que atualiza o saldo de modo que ContaCorente e ContaPoupança atualizem seus saldos com o dobro e o triplo da taxa fornecida, respectivamente.
6. Implemente uma classe **Ponto3d** para pontos tridimensionais (x,y,z) com construtores e operadores sobrecarregados que possibilitem a execução da função main() a seguir:

class Ponto3d{

int x; // Coordenada x do ponto

int y; // Coordenada y do ponto

int z; // Coordenada z do ponto

public:

...

};

int main() {

Ponto3d P1(2, -1,0), P2(1,4,-2), P3;

cout << P1; // mostra P1

cout << P2; // mostra P2

cout << P3; // mostra P3

P3 = P1 + P2; // soma dois pontos

cout << P3; // mostra P3

P3 = P1 – P2; // subtrai dois pontos

cout << P3; // mostra P3

P3 = P1 \* P2; // distância entre dois pontos

cout << p3; // mostra P3 (mudar)

if (P1 == P2) cout << “\npontos iguais ” ;

else cout << “\npontos diferentes” ;

cout << “Distancia entre P1 e P2 = “ << distancia (P1,P2);

return 0; }

Quando main() é compilada e executada, o seguinte resultado deve ser

produzido:

(2, -1, 0)

(1, 4, -2)

(0, 0, 0)

(3, 3, -2)

(1, -5, 2)

pontos diferentes

Distancia entre P1 e P2 = 5.47723

1. Faça um programa que leia uma string **s1**, uma string **s2**, um inteiro **pos** e insira a string **s2** em **s1** na posição **pos**.
2. Faça um programa que leia uma string **s** e apague todas as ocorrências de “ca” dessa string.
3. Faça um programa que insira os caracteres @@@ exatamente no meio de uma string.
4. Dado um inteiro **n**, use uma função recursiva para mostrar:

**n n-1 n-2 .... 1**

1. Uma lista simplesmente encadeada é um conjunto de nós, objetos criados dinamicamente na memória livre, em que cada nó da lista contém um membro de dados que se pretende armazenar e um ponteiro para o nó seu sucessor ou antecessor na lista. Implemente uma classe Node com a seguinte interface:

class Node {

public:

int data; // dado armazenado neste nó da lista

Node \*next; // ponteiro para antecessor da lista

Node(); // construtor para criar uma lista vazia

void insert(int n); // insere um objeto na lista, obtendo um novo nó, inserindo n em data e conectando o novo nó ao nodo antecessor. O primeiro nó da lista deve apontar para NULL(fim da lista).

bool remove(int n) ; // Caso exista na lista o inteiro n, remover o nó correspondente a n. Retornar true se removeu.

bool isEmpty(); // Retornar true se a lista está vazia

void display(); // Mostrar todos os elementos da lista

void removeAll(); // Remover todos os elementos da lista, devolvendo a memória ao espaço livre.

};

int main() {

Node N;

if (N.isEmpty()) cout << “lista vazia << endl;

else cout << “lista não vazia << endl;

N.insert(10);

N.insert(20);

N.insert(30);

N.display();

if (N.isEmpty()) cout << “lista vazia << endl;

else cout << “lista não vazia << endl;

if (N.remove(20)) cout << “Removido” << endl;

else cout << “Não removido << endl;

N.display();

N.removeAll();

if (N.isEmpty()) cout << “lista vazia << endl;

else cout << “lista não vazia << endl;

return 0; }

Quando main() é compilada e executada, o seguinte resultado deve ser

produzido:

lista vazia

10

20

30

lista não vazia

Removido

10

30

lista vazia