CIÊNCIA DA COMPUTAÇÃO
PROGRAMAÇÃO ORIENTADA A OBJETOS
PROF. JUDSON SANTIAGO

LABORATÓRIO 9

SOBRECARGA DE OPERADORES

EXERCÍCIOS DE REVISÃO

VOCÊ DEVE RESPONDER PARA REVISAR OS CONCEITOS IMPORTANTES

1. Sobrescreva os operadores adequados para fornecer à classe Tempo suporte a:

   - Adição
   - Subtração
   - Multiplicação por um fator
   - Adição de um fator

```
class Tempo
{
private:
    int horas;
    int minutos;

public:
    Tempo();
    Tempo(int h, int m = 0);

    void  Exibir() const;
    Tempo Somar(const Tempo& t) const;
};
```

Teste a classe com o seguinte programa:

```
int main()
{
    Tempo a { 2, 30 };
    Tempo b { 1, 10 };
    Tempo c { 0, 20 };

    Tempo total = a + b - c;
    total.Exibir();
    total = total + 2;
    total.Exibir();
    total = total * 2;
    total.Exibir();
};
```

EXERCÍCIOS DE FIXAÇÃO

VOCÊ DEVE FAZER OS EXERCÍCIOS PARA FIXAR O CONTEÚDO

1. Expanda a classe Tempo da questão anterior para que ela suporte também os operadores combinados += e -= para lidar com soma e subtração de Tempos. Observe que estes operadores, ao contrário da soma e subtração convencional, modificam o objeto no lado esquerdo da operação.

   Faça com que a classe funcione com o código abaixo:

```
int main()
{
    Tempo a { 2, 30 };
    Tempo b { 1, 10 };
    Tempo c { 0, 20 };

    Tempo total;
    total += a;
    total.Exibir();
    total += b;
    total.Exibir();
    total -= c;
    total.Exibir();
};
```

2. Crie uma Classe Packet para representar um pacote de informações a ser transferido pela rede. A classe deve fornecer uma interface amigável para agrupar 4 inteiros de 16 bits em um único inteiro de 64 bits.

```
int main()
{
    Packet packet;

    cout << "Empacotando..." << endl;
    packet.begin();
    packet << 9;
    packet << 4;
    packet << 3;
    packet << 7;
    packet.end();

    cout << "Enviando pacote..." << endl;
    packet.send();
    cout << "Recebendo pacote..." << endl;

    cout << "Desempacotando..." << endl;
    short a = 0, b = 0, c = 0, d = 0;
    packet >> a;
    packet >> b;
    packet >> c;
    packet >> d;
    cout << a << b << c << d << endl;
}
```

A união abaixo mostra como os dados devem ser guardados na classe Packet. Ela permite que os dados sejam acessados de forma individual, através do membro `part`, ou todo de uma vez, através do membro `all`.

```
union Data
{
    struct {
        short x;
        short y;
        short z;
        short w; } part;

    long long all;
};
```

Sobrescreva os operadores << e >> para que eles funcionem com a classe Packet. Forneça também um método send( ) para simular o envio do pacote pela rede. O método deve apenas exibir o valor de 64 bits armazenado no campo `all` da união.

3. Crie uma classe para representar uma lista de elementos, semelhante ao exemplo abaixo. A Lista deve armazenar os itens em um vetor dinâmico e possuir métodos para adicionar novos itens e verificar se a lista está vazia ou cheia.

```
class Lista
{
private:
    Item * itens;
    int size;
    int max;

public:
    Lista(int tam);
    ~Lista();

    bool Vazia() const;
    bool Cheia() const;
    bool Adicionar(const Item & item);
};
```

Sobrescreva a função `operator[]` para fornecer acesso direto a qualquer elemento da lista. O operador deve retornar o item na posição indicada ou deve retornar 0 e exibir uma mensagem de erro caso a lista não possua elementos no índice solicitado. Abaixo está um exemplo de uso da classe:

```
int main()
{
    Lista lista {5};
    lista.Adicionar(2); lista.Adicionar(3); lista.Adicionar(6);
    cout << lista[0];  // deve exibir 2
    cout << lista[3];  // deve exibir 0 e mensagem de erro
}
```

EXERCÍCIOS DE APRENDIZAGEM

VOCÊ DEVE ESCREVER PROGRAMAS PARA REALMENTE APRENDER

1. Partindo da classe Tempo, vista nos exercícios anteriores, sobrescreva o operador de comparação == para verificar se dois objetos da classe Tempo são iguais. A implementação deve permitir que o código abaixo funcione:

```
int main()
{
    Tempo a { 2, 30 };
    Tempo b { 1, 10 };
    Tempo c { 3, 40 };

    if (a == b) cout << "iguais" << endl;
    if (a + b == c) cout << "iguais" << endl;
};
```

2. Modifique a classe Packet, vista nos exercícios anteriores, para que seus dados possam ser modificados usando acesso direto, semelhante ao acesso feito em vetores. Para isso sobrescreva o operador [ ] de forma a permitir que o programa abaixo funcione corretamente.

```
int main()
{
    Packet packet;

    cout << "Empacotando..." << endl;
    packet[0] = 1;
    packet[1] = 2;
    packet[2] = 3;
    packet[3] = 4;

    cout << "Enviando pacote..." << endl;
    cout << "Recebendo pacote..." << endl;

    cout << "Desempacotando..." << endl;
    short a = 0, b = 0, c = 0, d = 0;
    a = packet[0];
    b = packet[1];
    c = packet[2];
    d = packet[3];
    cout << a << b << c << d << endl;
}
```

A função `operator[]` pode ter o seguinte protótipo:

```
short & operator[](int index);
```

Retornando uma referência é possível utilizar o operador tanto para recuperar um valor quanto para modificá-lo. A variável index receberá o valor entre colchetes.

3. Crie uma classe Packager que utilize um vetor dinâmico para armazenar uma sequência de valores inteiros de tamanho qualquer. A classe deve agir como uma empacotadora de números. Sua função é armazenar uma lista de números e fornecer um método para empacotar estes números em pacotes de 4.

   Ela deve ser usada como no exemplo abaixo:

```
int main()
{
    Packager packager {10};

    cout << "Adicionando números..." << endl;
    packager[0] = 9;
    packager[1] = 2;
    packager[2] = 8;
    packager[3] = 7;
    packager[4] = 1;
    packager[5] = 3;
    packager[6] = 5;
    packager[7] = 8;
    packager[8] = 1;
    packager[9] = 6;
    packager[10] = 4; // deve exibir mensagem de erro e
                      // nenhum valor deve ser armazenado

    cout << "Conteúdo da Empacotadora:" << endl;

    for (int i=0; i<10; ++i)
        cout << packager[i] << " ";

    cout << "Criando e enviando pacotes..." << endl;
    packager.send(); // deve exibir 9287
    packager.send(); // deve exibir 1358
    packager.send(); // deve exibir 1600
}
```

   A classe deve ter os seguintes recursos:

   - Os valores armazenados devem ser do tipo short
   - O construtor deve exigir o tamanho do pacote
   - Crie um vetor dinâmico com o tamanho fornecido no construtor
   - Libere o vetor dinâmico no destrutor
   - Forneça acesso aos elementos do vetor dinâmico por meio do operador [ ]
   - Realize a verificação dos índices antes de acessar o vetor dinâmico: se o índice solicitado não existir, exiba uma mensagem de alerta e não modifique o vetor dinâmico
   - Forneça um método `send()` para unir 4 valores short em um único valor long long. Esse método pode usar a classe Packet criada no Exercício de Fixação para simplificar o processo.
   - Faça com que o método `send()` exiba o valor empacotado
   - Cada chamada a `send()` deve empacotar o próximo grupo de 4 valores do vetor dinâmico. Se não existirem 4 números disponíveis, o pacote deve ser completado com valores zero.