

## INF 213 - Roteiro da Aula Prática 4

Arquivos fonte e diagramas utilizados nesta aula:

<https://drive.google.com/open?id=15ZsjG4QhYaTtI91mj8ci9r1t9MtDVBCe>

### Etapa 1

Implemente uma classe genérica *Conjunto* para representar um conjunto de elementos. O conjunto deve ser representado por um array, ALOCADO DINAMICAMENTE, cujo tamanho (isto é, o número máximo de elementos que o conjunto pode conter) deve ser definido pelo usuário na declaração do objeto usando o construtor; isto é, o tipo dos elementos deve ser um parâmetro da classe genérica e o tamanho deve ser um parâmetro para o construtor. O usuário pode optar por não definir este tamanho (ou seja, o tamanho deve ter um valor padrão) e neste caso, o conjunto deve ser criado com capacidade para 10 elementos. Veja o diagrama UML da classe em [Conjunto1UML.pdf](#).

Você deverá implementar pelo menos as funções abaixo (algumas funções extras serão necessárias). Elas devem ter o seguinte comportamento:

(a) `pertence(x)` : retorna o valor booleano *true* ou *false* para indicar respectivamente se o item *x* pertence ou não ao conjunto

(b) `insere(x)` : insere o elemento *x* no conjunto. Esta função deve retornar um valor booleano *true* ou *false* para indicar se a operação de inserção foi realizada com sucesso. Há duas possibilidades para que o elemento não possa ser inserido: caso o elemento a ser inserido já esteja no *Conjunto* (pois, conjuntos não podem ter elementos repetidos) ou a capacidade máxima do array foi alcançada.

(c) `numelementos()` : retorna o número de elementos no conjunto

(d) `operator ==` : verifica a igualdade; por exemplo, `A == B` retorna *true* ou *false* indicando respectivamente se os conjuntos *A* e *B* contêm os mesmos elementos.

(e) `operator <<` : envia para o stream de saída o conteúdo do conjunto; por exemplo, `cout << A`, imprime na tela os elementos do conjunto. Os valores devem ser impressos na ordem em que foram inseridos no conjunto. O conjunto deve ser impresso dentro de um par de colchetes e os valores devem estar separados por vírgula. (exemplo: `{1,9,2}` representa o conjunto com os números 1, 9 e 2).

(f) `operator >>` : extrai do stream de entrada o conteúdo do conjunto; por exemplo, `cin >> A`, lê do teclado os elementos do conjunto. A entrada deve ser lida enquanto o stream estiver válido (exemplo: se os dados forem lidos de um arquivo a leitura deverá parar no final do arquivo). Cada elemento da entrada será separado por um espaço em branco (diferentemente da saída, não haverá colchetes e vírgulas na entrada). Observe

que esta função (de leitura) também deve garantir que o conjunto não possui valores duplicados.

### **OBSERVAÇÕES IMPORTANTES:**

- Crie um arquivo **Conjunto1.h** para conter esta primeira versão da classe *Conjunto* .
- Lembre-se de usar const/referencia conforme necessario.
- Os operadores << e >> não serao testados apenas com cout/cin...

Teste a sua classe rodando o seguinte programa [TesteConjunto1.cpp](#). Para gerar o mesmo resultado listado no final do programa digite os valores indicados.

### **Etapa 2**

Crie um arquivo **Conjunto2.h** que estende a classe *Conjunto* incluindo as seguintes funções (veja o diagrama UML da classe em [Conjunto2UML.pdf](#)):

(i) operator + : operação de união; por exemplo  $A + B$  retorna o conjunto correspondente à união entre A e B. A capacidade (tam\_array) do novo conjunto devera ser a soma da capacidade das duas entradas. No conjunto resultante os elementos que estao em A deverao aparecer antes dos elementos (não repetidos) que estao em B.

(ii) operator \* : operação de interseção; por exemplo,  $A * B$  retorna o conjunto correspondente à interseção entre A e B. A capacidade (tam\_array) do novo conjunto devera ser igual a do conjunto com menor capacidade. A ordem dos elementos no conjunto resultante devera ser igual a ordem relativa deles em A.

(iii) operator - : operação de diferença: por  $A - B$  retorna o conjunto com o elementos que estão em A e não estão em B. A capacidade (tam\_array) do novo conjunto devera ser igual a capacidade de A. A ordem dos elementos no conjunto resultante devera ser igual a ordem relativa deles em A.

Teste a sua classe rodando [TesteConjunto2.cpp](#)

Observe a ordem esperada para os elementos do novo conjunto.

### **Submissao da aula pratica:**

A solucao deve ser submetida ate as 18 horas da proxima Segunda-Feira utilizando o sistema submittity ([submittity.dpi.ufv.br](http://submittity.dpi.ufv.br)). Envie apenas os arquivos Conjunto1.h e Conjunto2.h . Seu software sera testado considerando vários casos de teste (não haverá testes ocultos esta semana), incluindo testes que avaliam erros de memoria (se seu programa tiver algum erro de memoria haverá uma mensagem de erro na area “stderr” do submittity). Atualmente a submissao so pode ser realizada dentro da rede da UFV.

