INF 213 - Roteiro da Aula Pratica 4

Arquivos fonte e diagramas utilizados nesta aula:
https://drive.google.com/open?id=15ZsjG4QhYaTtl91mj8ci9r1t9MtDVBCe

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 devera implementar pelo menos as funcoes abaixo (algumas funcoes extras serao necessarias). 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 nao 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 contem os mesmos elementos.

(e) operador << : envia para o stream de saida 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 virgula. (exemplo: “{1,9,2}” representa o conjunto com os numeros 1, 9 e 2).

(f) operador >> : 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 valido (exemplo: se os dados forem lidos de um arquivo a leitura devera parar no final do arquivo). Cada elemento da entrada sera separado por um espaco em branco (diferentemente da saida, não havera colchetes e virgulas na entrada). Observe

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

OBSERVAÇOES 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***). Dica REUSE funções que você já criou!

(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.

Etapa 3

Crie um arquivo chamado Respostas.txt e escreva nele as respostas para esta estapa (envie-as pelo submitty).

1) Qual a complexidade de tempo das seguintes funcoes/operadores: pertence(), insere(), numElementos(), +, *, - .

2) Considerando a função “+”, a ordem de complexidade do algoritmo depende da forma com que os elementos estão ordenados nos conjuntos? Justifique.

3) Crie um programa que constrói dois conjuntos (do tipo Conjunto2) de inteiros A, B (cada um contendo *n* elementos, sendo que A = {1,2,3, …, n} e B={1,2,3, …, n}) e, a seguir, calcule a união entre A e B utilizando o operador “+” (guarde o resultado disso em C). Por fim, imprima o resultado de “C.pertence(10)” na tela (não imprima o conjunto C… apenas o resultado dessa chamada -- isso deve ser feito para evitar que o compilador veja que nada está sendo feito com C e, assim, otimize o código removendo o cálculo de C). Meça o tempo de execuçao do seu programa para os seguintes valores de n: 10,100, 1000, 5000,10000, 20000,40000 . Escreva os tempos obtidos e o que você pode concluir a partir deles. Salve seu programa com o nome p31.cpp (e submeta junto com seu exercício).

4) Repita os testes acima supondo que, dessa vez, A = {1,2,3, …, n} e B={-1,-2,-3,-4,...,-n}. Houve diferença no tempo? Houve uma diferença na forma com que o tempo cresce? Por que isso ocorre? Salve seu programa com o nome p32.cpp (e submeta junto com seu exercício).

5) Repita os testes anteriores do item 4, mas agora compilando seu programa usando o comando “g++ -O3 programa.cpp”. A flag O3 ativa o nível de otimização máximo (o compilador utilizará muitas técnicas para melhorar o desempenho do seu programa). O que você observou em relação aos tempos obtidos?

Dica: para medir o tempo de execução de um programa no Linux, use o comando “time ./a.out” (onde a.out é o programa cujo tempo está sendo medido). Considere o tempo “real” retornado pelo time.

Submissao da aula pratica:
A solucao deve ser submetida ate as 18 horas da proxima Segunda-Feira utilizando o sistema submitty (submitty.dpi.ufv.br). Envie apenas os arquivos Conjunto1.h e Conjunto2.h . Seu software sera testado considerando vários casos de teste (não havera testes ocultos esta semana), incluindo testes que avaliam erros de memoria (se seu programa tiver algum erro de memoria havera uma mensagem de erro na area “stderr” do submitty).