Trabalho escrito 3

segunda-feira, 15 de maio de 2023 00:24

Leonardo O Campos.

LISTA DE EXERCÍCIOS

REFERÊNCIAS RVALUE, SEMÂNTICA DE TRANSFERÊNCIA E ENCAMINHAMENTO PERFEITO

Questão 01.
O que é a técnica *Small String Optimization (SSO)* utilizada em algumas implementações de linguagens de programação, como C++? Quais as vantagens e desvantagens dessa técnica?

É uma técnica utilizada em algumas linguagens de programação para otimizar o armazenamento de strings curtas. Basicamente, armazenando a string diretamente no objeto que a contém, não sendo necessário alocação de memória de forma dinâmica. Esse feito é possível visto que no processo de compilação é feito o cálculo do tamanho da string, assim pode ser reservado o tamanho no objeto.

Vantagens	Desvantagens
- economia de memória - melhora o desempenho - Armazena direto no obj - Reduz a fragmentação de memória.	- Só é eficaz para strings curtas. - Aumenta o obj se a string for grande.

Small String Optimization (SSO)

Questão 02.
O que é a otimização *Named Return Value Optimization (NRVO)*?

Técnica usada para otimizar o desempenho de funções que retornam objetos. A ideia é evitar cópia desnecessária de objetos retornados por valor, permitindo que o compilador otimize o código para evitar a cópia.

Questão 03.
O que é a técnica "*copy and swap*" e quais são as suas vantagens e desvantagens? Escreva um código de uma classe qualquer que implemente essa técnica?

Usado para implementar o operador de atribuição de uma classe. Cria uma cópia da classe e troca os valores da cópia com os valores da classe original. Em seguida, a cópia é destruída e o operador de atribuição retorna um ponteiro para a classe original

Vantagens: - melhora de desempenho.
- evita erros de gerenciamento de memória.

→ Código em anexo.

Questão 04.
A função `std::sort` da STL faz uso da função `swap` para realizar trocas entre os elementos de um container (por exemplo, `std::vector`) durante a ordenação. Isso nos leva a pensar que, ao ordenar um container, não serão criadas cópias temporárias caso exista uma definição especializada de `swap` para o tipo dos objetos armazenados

Questão 04.
A função `std::sort` da STL faz uso da função `swap` para realizar trocas entre os elementos de um container (por exemplo, `std::vector`) durante a ordenação. Isso nos leva a pensar que, ao ordenar um container, não serão criadas cópias temporárias caso exista uma definição especializada de `swap` para o tipo dos objetos armazenados no container. Porém, isso não é verdade. a função `std::sort` faz algumas cópias ao invés de trocas.

(a) Escreva um código para testar essa situação.
(b) Explique o porquê desse comportamento de `std::sort`.

a) → Código em anexo!

b) Isso ocorre porque a função "sort" é implementada usando o algoritmo "quicksort", que é um algoritmo de ordenação baseado em comparação. O "quicksort" funciona dividindo o array em duas partes e ordenando recursivamente essas partes. Durante a ordenação, os elementos são comparados e trocados de posição se necessário. No entanto, em alguns casos, a troca não é possível e uma cópia é feita em vez disso.

Questão 05.
O que significa a transferência de um tipo primitivo?

Um tipo primitivo é um tipo de dado básico que é suportado diretamente pelo compilador. No C++, temos inteiros, float e caracteres. Quando falamos de transferência de um tipo primitivo estamos nos referindo à passagem de um valor de uma variável de um tipo primitivo para outra variável do mesmo tipo primitivo.

Questão 06 (prática).
Implemente uma classe com nome `Racional`, a qual será utilizada para representar números racionais de maneira exata. Para isso, a ele deverá representar os valores em um formato de fração irredutível, ou seja, internamente a classe deverá ter um atributo inteiro para representar o numerador e outro atributo para representar o denominador. Caso o valor representado seja negativo, o numerador é quem deverá ser negativo. Além disso, a classe `Racional` deverá implementar:

- Construtor padrão (construtor que não recebe nenhum argumento), o qual deverá produzir um objeto de `Racional` que represente o valor zero.
- Construtor que receba dois números inteiros sendo, respectivamente, o numerador e o denominador.
- Construtor de cópia (construtor que recebe uma referência de *lvalue* constante para o mesmo tipo da classe).
- Construtor de transferência (construtor que recebe uma referência de *rvalue* para o mesmo tipo da classe).
- Destrutor, caso seja necessário desalocar algum recurso dinâmico alocado pela classe.
- Operadores de atribuição por cópia e por transferência.
- Operadores relacionais `<`, `<=`, `>`, `>=`, e `==`. Tais operadores devem ser métodos constantes, pois não modificam o objeto.
- Operadores aritméticos `+`, `-`, `*`, `/`, `+=`, `-=`, `*=`, `/=`, `++` (pré-fixado e pós-fixado), `--` (pré-fixado e pós-fixado), e o operador `-` (unário). Os operadores aritméticos que não modificam o objeto devem ser implementados como métodos constantes.
- Métodos constantes `numerador()` e `denominador()` que retornem referências constantes para o numerador e denominador, respectivamente.
- Métodos (não constantes) `numerador(int num)` e `denominador(int den)` que alterem os valores do numerador e do denominador, respectivamente.
- Um método constante `valor()` que retorne um valor do tipo `double` referente ao valor representado pelo objeto de Racional.

Após a implementação da classe `Racional`, crie um `std::vector<Racional>` com alguns objetos de `Racional` e em seguida, ordene esse vector usando o método `std::sort` da STL. A partir dos valores presentes no vector, calcule o somatório, gerando um objeto do tipo `Racional` como resultado.

→ Código em anexo!