Armazenamento de dados em memória (revisão de ponteiros e alocação dinâmica), Listas encadeadas

1. Para cada um dos itens seguintes, escreva uma única instrução que realiza a tarefa indicada. Suponha que as variáveis do tipo inteiro `long value1` e `value2` tenham sido declaradas e que value1 tenha sido inicializado como 200000.
   (a) Declare a variável `longPtr` como um ponteiro para um objeto do tipo `long`.
   (b) Atribua o endereço da variável `value1` à variável ponteiro `longPtr`.
   (c) Imprima o valor do objeto apontado por `longPtr`.
   (d) Atribua o valor do objeto apontado por `longPtr` à variável `value2`.
   (e) Imprima o valor de `value2`.
   (f) Imprima o endereço de `value1`.
   (g) Imprima o endereço armazenado em `longPtr` . O valor impresso é o mesmo que o endereço de `value1`?
2. Para cada um dos itens a seguir, escreva instruções C++ que realizam a tarefa especificada. Suponha que inteiros sem sinal estejam armazenados em dois bytes e que o endereço inicial do array esteja na posição 1002500 da memória.
   (a) Declare um array do tipo `unsigned int` chamado `values` com cinco elementos e inicialize os elementos para os inteiros pares de 2 a 10. Suponha que a constante simbólica `SIZE` foi definida como 5.
   (b) Declare um ponteiro `vPtr` que aponta para um objeto do tipo `unsigned int`.
   (c) Utilize uma instrução `for` para imprimir os elementos do array `values` usando notação de array subscrito.
   (d) Escreva duas instruções separadas que atribuem o endereço inicial do array values à variável ponteiro `vPtr`.
   (e) Utilize uma instrução `for` para imprimir os elementos do array `values` utilizando a notação de ponteiro/deslocamento.
   (f) Utilize uma instrução `for` para imprimir os elementos do array `values` utilizando a notação de ponteiro/deslocamento com o nome de array como o ponteiro.
   (g) Utilize uma instrução for para imprimir os elementos do array `values` utilizando subscritos no ponteiro para o array.
   (h) Referencie o quinto elemento de `values` utilizando a notação de subscrito de array, a notação de ponteiro/deslocamento com o nome de array como o ponteiro, a notação de subscrito de ponteiro e a notação de ponteiro/deslocamento.
   (i) Mostre que endereço é referenciado por `vPtr + 3`, e que valor é armazenado nessa localização.
   (j) Supondo que `vPtr` aponte para `values[4]`, que endereço é referenciado por `vPtr -= 4`? Que valor é armazenado nessa localização?
3. Implemente um código para remover duplicatas de uma lista encadeada não ordenada.
4. Implemente um código para encontrar o k-ésimo elemento de uma lista encadeada.
5. Escreve um código para particionar uma lista encadeada em volta de um valor `x`, tal que todos os nós menores que `x` venham antes de todos os nós maiores que ou iguais à `x`. Se `x` estiver contido dentro da lista, os valores de `x` só precisam vir depois dos elementos menores do que `x`. O elemento `x` pode aparecer em qualquer posição na partição direita, ele não precisa aparecer entre as partições esquerda e direita.

6. Implemente uma função para checar se uma lista duplamente encadeada é um palíndromo.
7. Dado uma lista ordenada de inteiros distintos, escreva um algoritmo para criar uma árvore binária de pesquisa com altura mínima.
8. Escreva um código para encontrar o "próximo" nó (sucessor em-ordem) de um dado nó em uma árvore binária de pesquisa. Assuma que cada nó tem um ponteiro para seu pai.
9. Escreva uma função para encontrar o primeiro ancestral comum de dois nós em uma árvore binária.
10. Você está implementando uma classe de árvore binária que, além das funções `insere`, `busca`, e `deleta`, tem um método `getNoAleatorio` que retorna um nó aleatório da árvore. Todos os nós possuem a mesma probabilidade de serem escolhidos. Implemente o código para a função `getNoAleatorio`.

Depuração

1. Localize o erro em cada um dos seguintes segmentos de programa e explique como o erro pode ser corrigido.

(a)
```
int g( void ) {
  cout << "Dentro da funcao g" << endl;
  int h( void ) {
    cout << "Dentro da funcao h" << endl;
  }
}
```

(b)
```
int sum( int x , int y ) {
  int result;
  result = x + y;
}
```

(c)
```
int soma( int n ) {
  if ( n == 0 )
    return 0;
  else
    n + soma( n - 1 );
}
```

(d)
```
void product( void ) {
  int a;
  int b;
  int c;
  int result;
  cout << "Digite tres inteiros: ";
  cin >> a >> b >> c;
  result = a * b * c;
  cout << "Resulado: " << result;
  return result;
}
```

2. O que o seguinte programa faz?

```
#include <iostream>
using std::cout;
using std::endl;

int misterio( int [], int );

int main() {
  const int arraySize = 10;
  int a[ arraySize ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  int result = misterio( a, arraySize );
  cout << "Resultado: " << result << endl;
  return 0;
}

// O que essa funcao faz?
int misterio( int b[], int size ) {
  if ( size == 1 )
    return b[ 0 ];
  else
    return b[ size - 1 ] + misterio( b, size - 1 );
}
```

3. Encontre o erro em cada um dos seguintes segmentos de programa. Suponha as seguintes declarações e instruções:

```
int *zPtr; // zPtr ira referenciar o array z
int *aPtr = 0;
void *sPtr = 0;
int number;
int z[ 5 ] = { 1, 2, 3, 4, 5 };
```

(a)
```
++zPtr;
```

(b)
```
number = zPtr;
```

(c)
```
number = *zPtr[ 2 ];
```

(d)
```
for ( int i = 0; i <= 5; i++ )
  cout << zPtr[ i ] << endl;
```

(e)
```
number = *sPtr;
```

4. Encontre o(s) erro(s) do seguinte programa.

```
#include <iostream>
#include <string>

int main () {
  std::string str1 ("laranja");
  std::string str2 ("laranja");
  if ( str1.compare(str2) )
    cout << "As strings sao iguais" << endl;
  return 0;
}
```

5. Localize o(s) erro(s) em cada uma das seguintes sequências e explique como corrigi-lo(s):

(a)
```
void ~Tempo( int );
```

(b) A seguinte definição é uma definição parcial da classe Tempo:

```
struct Tempo
{
          // prototipos de funcao
          int hora = 0;
          int minuto = 0;
          int segundo = 0;
}; // fim da classe Time
```

6. Localize os erros na seguinte classe e explique como corrigi-los:

```
struct Exemplo {
    Exemplo( int y = 10 ) : dado( y ) {}
    int getDadoIncrementado() const {
      return dado++;
    }
    static int getContador() {
      return contador;
    }
    int dado;
    static int contador;
};
```

Programação Orientada a Objetos

1. Crie uma classe `Data` com três atributos inteiros: `dia`, `mês` e `ano`. Faça um construtor que inicializa as três variáveis e suponha que os valores passados serão corretos. A classe deve possuir um método para exibir a data em formato de números separados por barra: dia/mes/ano e outro método para exibir a data por extenso (ex: 12 de janeiro de 2015).

2. Crie uma classe `Rectangle` com atributos `length` e `width`, cada um dos quais assume o padrão de 1. Forneça funções-membro que calculam os atributos `perimeter` e `area` do retângulo. Além disso, forneça as funções `set` e `get` para os atributos `length` e `width`. As funções `set` devem verificar se `length` e `width` são números de ponto flutuante maiores que $0,0$ e menores que $20,0$.

3. Implemente em C++ uma classe chamada `Aquecedor`. Ela deve ter um único atributo chamado `temperatura`, cujo tipo deve ser um ponto flutuante de precisão dupla. Defina um construtor que não recebe parâmetros e inicializa a temperatura em 20 graus. Crie os métodos aquecer e resfriar que aumentam e diminuem a temperatura em 5 graus, respectivamente. Defina um método para retornar o valor da temperatura.

4. Altere a classe do exercício anterior para que ela tenha três novos atributos: temperatura mínima, temperatura máxima e fator de incremento da temperatura. Os dois primeiros devem ser inicializados com 10 e 40 graus respectivamente no construtor. A classe deve ter um construtor sem parâmetros, que definirá o fator de incremento em 5 graus, um segundo construtor que recebe a temperatura inicial e um terceiro que recebe a temperatura inicial e o fator de incremento.

   Altere os métodos existentes na classe de forma apropriada com o objetivo de manter o estado do objeto sempre válido (ex: o fator de incremento deve ser usado toda vez que os métodos aquecer e resfriar forem chamados). Escreva mensagens na saída padrão quando uma ação não puder ser executada por não ser um estado de objeto válido.

   Por fim, crie um método que permita alterar o fator de incremento da temperatura depois de um objeto já ter sido criado.

5. Crie uma classe `SavingsAccount`. Utilize um membro de dados `static annualInterestRate` para armazenar a taxa de juros anual para cada um dos correntistas. Cada membro da classe contém um membro de dados `private savingsBalance` para indicar a quantia que os correntistas têm atualmente em depósito. Forneça a função-membro `calculateMonthlyInterest` que calcula os juros mensais multiplicando o `balance` [saldo] pelo `annualInterestRate` dividido por 12; esses juros devem ser adicionados a `savingsBalance`. Forneça uma função-membro `static modifyInterestRate` que configura o `static annualInterestRate` com um novo valor. Escreva um programa de driver para testar a classe `SavingsAccount`. Instancie dois objetos diferentes da classe `SavingsAccount`, `saver1` e `saver2`, com saldos de $\$\ 2.000,00$ e $\$\ 3.000,00$, respectivamente. Configure o `annualInterestRate` como 3%. Em seguida, calcule os juros mensais e imprima os novos saldos de cada um dos correntistas. Então configure o `annualInterestRate` como 4%, calcule os juros do próximo mês e imprima os novos saldos para cada um dos poupadores.

Encapsulamento, Herança e Composição

1. Desenhe uma hierarquia de herança para alunos universitários. Utilize Aluno como a classe básica da hierarquia, então inclua as classes AlunoDeGraduação e AlunoGraduado que derivam de Aluno. Continue a estender a hierarquia o mais profundamente (isto é, com muitos níveis) possível. Por exemplo, Primeiranistas, Segundanistas, Terceiranistas e Quartanistas poderiam derivar de AlunoDeGraduação; e AlunoDeDoutorado e AlunoDeMestrado poderiam derivar de AlunoGraduado. Depois de desenhar a hierarquia, discuta os relacionamentos entre as classes.

2. Os serviços de correio expresso, como FedEx, DHLe UPS, oferecem várias opções de entrega, cada qual com custos específicos. Crie uma hierarquia de herança para representar vários tipos de pacotes. Utilize `Package` como a classe básica da hierarquia, então inclua as classes `TwoDayPackage` e `OvernightPackage` que derivam de `Package`. A classe básica `Package` deve incluir membros de dados que representam nome, endereço, cidade, estado e CEP tanto do remetente como do destinatário do pacote, além dos membros de dados que armazenam o peso (em quilos) e o custo por quilo para a entrega do pacote. O construtor `Package` deve inicializar esses membros de dados. Assegure que o peso e o custo por quilo contenham valores positivos. `Package` deve fornecer uma função-membro `public calculateCost` que retorna um `double` indicando o custo associado com a entrega do pacote. A função `calculateCost` de `Package` deve determinar o custo multiplicando o peso pelo custo (em quilos). A classe derivada `TwoDayPackage` deve herdar a funcionalidade da classe básica `Package`, mas também incluir um membro de dados que representa uma taxa fixa que a empresa de entrega cobra pelo serviço de entrega de dois dias. O construtor `TwoDayPackage` deve receber um valor para inicializar esse membro de dados. `TwoDayPackage` deve redefinir a função-membro `calculateCost` para que ela calcule o custo de entrega adicionando a taxa fixa ao custo baseado em peso calculado pela função `calculateCost` da classe básica Package. A classe `OvernightPackage` deve herdar diretamente da classe `Package` e conter um membro de dados adicional para representar uma taxa adicional por quilo cobrado pelo serviço de entrega noturno. `OvernightPackage` deve redefinir a função-membro `calculateCost` para que ela acrescente a taxa adicional por quilo ao custo-padrão por quilo antes de calcular o custo da entrega. Escreva um programa de teste que cria objetos de todos os tipos de `Package` e testa a função-membro `calculateCost`.

3. Use a hierarquia de herança `Package` criada no exercício anterior para criar um programa que exibe asinformações de endereço e calcula os custos de entrega de vários `Package`s. O programa deve conter um vector de ponteiros `Package` para objetos das classes `TwoDayPackage` e `OvernightPackage`. Faça um loop pelo `vector` para processar o `Package`s polimorficamente. Para cada `Package`, invoque as funções `get` para obter as informações de endereço do remetente e do destinatário, e então imprima os dois endereços da maneira que apareceriam nos pacotes de correio. Além disso, chame a função-membro `calculateCost` de cada `Package` e imprima o resultado. Monitore o custo de entrega total de todos os `Package`s no `vector` e exiba esse total quando o loop terminar.

4. O mundo das formas é muito rico. Anote todas as formas que puder imaginar — bidimensionais e tridimensionais — e as forme em uma hierarquia `Forma` com o maior número de níveis possível que imaginar. Sua hierarquia deve ter a classe básica `Forma` a partir da qual a classe `FormaBiDimensional` e a `FormaTriDimensional` são derivadas. Implemente a hierarquia `Forma` projetada anteriormente. Cada `FormaBidimensional` deve conter a função `obterArea` para calcular a área da forma bidimensional. Cada `FormaTridimensional` deve ter funções-membro `obterArea` e `obterVolume` para calcular a área do volume e da superfície, respectivamente, da forma tridimensional. Crie um programa que utilize um vector de ponteiros `Forma` para objetos de cada classe concreta na hierarquia. O programa deve imprimir o objeto para o qual cada elemento `vector` aponta. Além disso, no loop que processa todas as formas no `vector`, determine se cada forma é uma `FormaBidimensional` ou `FormaTridimensional`. Se uma forma for uma `FormaBidimensional`, exiba sua área. Se uma forma for uma `FormaTridimensional`, exiba sua área e volume.

5. Crie uma hierarquia de herança que um banco possa utilizar para representar as contas bancárias dos clientes. Todos os clientes nesse banco podem depositar (isto é, creditar) dinheiro em suas contas e

retirar (isto é, debitar) o dinheiro delas. Há também tipos mais específicos de contas. As contas de poupança, por exemplo, recebem juros pelo dinheiro depositado nelas. As contas bancárias, por outro lado, cobram uma taxa por transação (isto é, crédito ou débito).

Crie uma hierarquia de herança contendo classe básica `Account` e classes derivadas `SavingsAccount` e `CheckingAccount` que herdam da classe `Account`. A classe básica `Account` deve incluir um membro de dados do tipo `double` para representar o saldo da conta. A classe deve fornecer um construtor que recebe um saldo inicial e o utiliza para inicializar o membro de dados. O construtor deve validar o saldo inicial para assegurar que ele é maior que ou igual a 0.0. Caso contrário, o saldo deve ser configurado como 0.0 e o construtor deve exibir uma mensagem de erro, indicando que o saldo inicial era inválido. A classe deve fornecer três funções-membro. A função-membro `credit` deve adicionar uma quantia ao saldo atual. A função-membro `debit` deve retirar dinheiro de `Account` e assegurar que o valor do débito não exceda o saldo de `Account`. Se exceder, o saldo deve permanecer inalterado e a função deve imprimir a mensagem "Debit amount exceeded account balance" [Saldo insuficiente]. A função-membro `getBalance` deve retornar o saldo atual.

A classe derivada `SavingsAccount` deve herdar a funcionalidade de uma `Account`, mas também incluir um membro de dados do tipo `double` para indicar a taxa de juros (porcentagem) atribuída à `Account`. O construtor `SavingsAccount` deve receber o saldo inicial, bem como um valor inicial para a taxa de juros de `SavingsAccount`. `SavingsAccount` deve fornecer uma função-membro `public calculateInterest` que retorna um `double` para indicar os juros auferidos por uma conta. A função-membro `calculateInterest` deve determinar esse valor multiplicando a taxa de juros pelo saldo da conta. Nota: `SavingsAccount` deve herdar as funções-membro `credit` e `debit` exatamente como são sem redefini-las.

A classe derivada `CheckingAccount` deve herdar da classe básica `Account` e incluir um membro adicional de dados do tipo `double` que representa a taxa cobrada por transação. O construtor `CheckingAccount` deve receber o saldo inicial, bem como um parâmetro que indica o valor de uma taxa. A classe `CheckingAccount` deve redefinir as funções-membro `credit` e `debit` para que subtraiam a taxa do saldo da conta sempre que qualquer uma das transações for realizada com sucesso. As versões `CheckingAccount` dessas funções devem invocar a versão `Account` da classe básica para realizar as atualizações de saldo de uma conta. A função `debit` de `CheckingAccount` deve cobrar uma taxa somente se o dinheiro for realmente retirado (isto é, o valor do débito não exceder ao do saldo da conta). Dica: Defina a função `debit` de `Account` para que ela retorne um `bool` indicando se houve retirada de dinheiro. Em seguida, utilize o valor de retorno para determinar se uma taxa deve ser cobrada.

Depois de definir as classes nessa hierarquia, escreva um programa que cria objetos de cada classe e testa suas funções-membro. Adicione os juros ao objeto `SavingsAccount` invocando primeiro sua função `calculateInterest` e, então, passando o valor retornado dos juros para a função `credit` do objeto.