| | |
|---|---|
| **Iniciada** | sexta, 12 de fevereiro de 2021 às 10:51 |
| **Estado** | Terminada |
| **Terminada em** | sexta, 12 de fevereiro de 2021 às 11:31 |
| **Tempo gasto** | 40 minutos 1 segundo |

**Pergunta 1**
Respondida
Nota: 20,00

Pretende-se implementar o conceito de Sequências de Valores que representam números naturais por uma determinada ordem. Os valores que vão surgindo na sequência dependem de uma regra simples de progressão. Por exemplo, cada valor pode ser obtido somando um valor constante ao anterior (sequência aritmética), ou multiplicando o valor anterior por uma constante (sequência geométrica), ou ainda outros tipos de sequência, que até podem depender não só do valor anterior mas de outros ainda mais para trás na sequência (por exemplo, a sequência de Fibonacci em que para $x_n = x_{n-1} + x_{n-2}$).

Neste exercício, pretende-se focar nas sequências aritméticas e geométricas, mas a solução apresentada deverá permitir acrescentar novas sequências específicas que progridem de outras formas.

Todas as sequências possuem um conjunto de valores (inicialmente vazio), um valor inicial (o primeiro da sequência) e uma quantidade de valores a calcular. A sequência aritmética possui um valor constante que deve ser somado ao valor atual para obter o seguinte, e a sequência geométrica tem um fator que é multiplicado pelo valor atual para calcular o seguinte. Assim uma sequência genericamente tem os seguintes dados (pode acrescentar novos e remover apenas 1 atributo - o tipo):

```
class Sequencia {
private:          // os dados devem continuar privados
   int valor_inicial, quantos;
   string tipo;   // único atributo que deve ser removido, não deve ser usado
   ... numeros;    // representa o conjunto de nº gerados através da função de cálculo
   ...             // da sequência

public:
   ...
};
...
int main(){

  //Sequencia seq(1, 10);  // Sequência de tipo indefinido, logo não aceita este objecto
  Aritmetica ari(1, 6, 2); // Seq. aritmética. Começa em 1, 6 números, incremento de 2

  Geometrica geo(5, 4, 3); //Seq. geométrica. Começa em 5, 4 números, fator multipl.de 3
```

```
  vector<Sequencia *> seqs;
  seqs.push_back(&ari);
  seqs.push_back(&geo);
  // Nas linhas seguintes usa-se a seq. aritmética, mas a geométrica é semelhante e
  // também deve funcionar
  seqs[0]->calcula();  // calcula e armazena os valores da sequência
  cout << seqs[0]->getElementoEm(4);  // Apresenta o 5º da sequência já calculada
  cout << seqs[0]->getElementoEm(15); // elemento não existe, retorna -1
  cout << seqs[0]->getAsString();     // mostra: "aritmética 1  3  5  7  9  11"
  // seq. geométrica em seqs[1] seria semelhante
}
```

Faça com que o código acima compile e funcione corretamente. A solução deve permitir representar corretamente os conceitos de sequências aritméticas e geométricas, obedecendo a todas as boas práticas da programação orientada a objetos (Abstração, Encapsulamento, Herança, Polimorfismo).

```
class Sequencia{
protected:
int valor_inicial, quantos;
int incremento;
// string tipo;


public:
Sequencia(int vi, int q): valor_inicial(vi), quantos(q){};

virtual void calcula()=0;
virtual string getAsString()=0;
virtual int getElemento( int pos )=0;
};
class Aritmetica : public Sequencia{
private:
int incremento;
vector <int > va;

public:
Aritmetica(int i, int vi, int q): incremento(i), Sequencia(vi,q){
};
void calcula() {
int i=0, novoNumero = 0;
while (i < quantos) {
if (va.size() == 0)
novoNumero = valor_inicial;
else
novoNumero == novoNumero + incremento;
va.push_back(novoNumero);
```

```
vi.push_back(novoNumero);
i++;



}

};

string getAsString() {
ostringstream os;
os << "\naritmetica ";
for (auto a : va)
os << " " << a;

return os.str();

};
int getElemento(int pos) {

for (auto a : va) {

if (a = pos) {
return va.data();
}
}
return -1;
};

};

class Geometrica : public Sequencia{
private:
int multiplica;
vector <int > vg;
public:
Geometrica(int m, int vi, int q): multiplica(m),Sequencia(vi,q){

};

void calcula(){
int i=0, novoNumero=0;

while( i < quantos ){
if ( vg.size() == 0 )
novoNumero = valor_inicial;
else
novoNumero == novoNumero * multiplica;

vg.push_back(novoNumero);
```

```
i++;
}



};


string getAsString(){
ostringstream os;
os << "\ngeometrica ";
for( auto a: vg )
os << " " << a;

return os.str();
};

int getElemento(int pos) {

for (auto a : vg) {

if (a = pos) {
return vg.data(a);
}
}
return -1;
};

};
```

**Pergunta 2**

Respondida

Nota: 15,00

Considere a seguinte classe:

```
class XYZ {
   vector<BC*> ptrs;
public:
   // ...
};
```

A classe XYZ tem uma relação de composição com os objetos que são armazenados no vector ptrs. Assuma que esta classe tem um método que adiciona ponteiros para objetos da classe BC e suas classes derivadas, sendo a alocação destes objetos feita em memória dinâmica. Assuma também que o destrutor da classe XYZ já está implementado. A classe BC tem duas classes derivadas: DC1 e DC2.

a) Tendo em conta a referida relação de composição, implemente o construtor por cópia e o operador de atribuição da classe XYZ. Pode adicionar código às classes XYZ, BC, DC1, e DC2, mas não pode alterar o código fornecido. Soluções com repetição desnecessária de código serão penalizadas. Justifique a sua implementação. Uma implementação correta sem a justificação associada vale 0.

b) Considere agora que a referida relação de composição passa a ser uma relação de agregação. Implemente o construtor por cópia e o operador de atribuição da classe XYZ, justificando a sua implementação. Uma implementação correta sem a justificação associada vale 0.

```
class BC{
};
class CD1 : public BC{
};
class DC2 : public BC{
}

class XYZ{
vector (BC*) ptrs;
public:
```

```
};
```

« PREVIOUS ACTIVITY
Exame de Recurso parte 1 de 3

NEXT ACTIVITY
Exam 2nd call - Part 1 of 3 »