UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO E ENGENHARIA DE COMPUTAÇÃO

LEONARDO DALCIN
PAULO RICARDO RAMOS DA ROSA
PIETRO DEGRAZIA

Jogo Pedagógico em C++

Relatório apresentado como requisito parcial para a obtenção de conceito na Disciplina de Modelos de Linguagens de Programação

Prof. Dr. Lucas Mello Schnorr
Orientador

Porto Alegre
2018

SUMÁRIO

1 INTRODUÇÃO 3
1.1 Projeto 3
1.2 Sobre o jogo 3
2 C++17 4
2.1 Origem 4
2.2 Programa 4
2.3 Objetos 4
2.4 Classes 5
2.5 Classes Abstratas 5
2.6 Polimorfismo 6
2.7 Memória 6
3 IMPLEMENTAÇÃO 7
3.1 Orientação a Objetos 7
3.2 Recursos utilizados para a implementação OO 7
3.3 Funcional 9
3.4 Recursos utilizados para a implementação Funcional 9
3.5 Referências 10

1 INTRODUÇÃO

Este trabalho tem como objetivo a implementação em C++ de um jogo pedagógico que visa ensinar diversos conceitos da cadeira de Modelos de Linguagens de Programação em diferentes níveis de profundidade. O programa permitirá que o usuário compartilhe seus resultados e confira respostas de outros colegas, assim como explicações e links para referências sobre o assunto.

1.1 Projeto

A proposta do projeto consta em, dada uma linguagem de programação escolhida pelo grupo dentre as pré-selecionadas pelo professor, desenvolver uma aplicação com duas implementações: uma puramente orientada a objetos e outra puramente funcional, contendo alguma forma de paralelismo. C++17 será a linguagem utilizada para ambas as versões.

1.2 Sobre o jogo

O projeto será um jogo de perguntas e respostas sobre a disciplina de Modelos de Linguagem de Programação, aos moldes dos programas "Show do Milhão"e "Quem quer ser um milionário?", com a objetivo de ajudar o estudante na disciplina. O jogo contará com explicações sobre o conteúdo das perguntas e poderá ser feito a consulta das respostas dos outros jogadores.

2 C++17

2.1 Origem

A linguagem C++ foi introduzida inicialmente em 1985 por Bjarne Stroustrup, herdando as características da linguagem C mas adicionando o conceito de Orientação a Objetos. Desde então, passou por diversas atualizações, estando hoje padronizada na versão ISO/IEC 14882:2017, informalmente conhecida como C++17.

2.2 Programa

Programas em C++ podem conter valores, objetos, referências, funções, enumeradores, tipos, membros de classes, templates, especializações de template e namespaces. Essas entidades são criadas através de declarações, que associam a entidade com um nome e define suas propriedades. A declaração que define todas as propriedades para usar uma entidade é chamada de definição.

Nomes encontrados em um programa são associados com a declaração em que aparecem. Eles somente são válidos no escopo em que foram declarados. Alguns nomes, dependendo da linkagem, podem referênciar entidades que aparecem em um escopo diferente.

2.3 Objetos

Programas em C++ criam, destroem, referenciam, acessam e manipulam objetos. Um objeto em C++ é uma região de armazenamento que possui:

- tamanho(sizeof);
- requisito de alinhamento(alignof);
- duração de alocação(automática, estática, dinâmica, local);
- duração de vida(baseada na alocação ou temporária);
- tipo;
- valor(pode ser indeterminado);

- nome(opcional).

Objetos são criados com definições, expressões new, expressões throw, por manipulação de membros de uma union e onde objetos temporários são necessários.

2.4 Classes

Classes e estruturas são tipos definidos pelo usuário, usando especificadores de classe, que devem aparecer na sequencia a seguir.

- Chave de classe - pode ser class ou struct, a única diferença entre elas é o acesso padrão, para estruturas é acesso a membro, e para classes é a membro da classe base.

- Atributos - sequencia opcional de atributos, noreturn, por exemplo. Também pode ser incluido especificadores de alinhamento.

- Nome - o identificador da classe. Opcionalmente acompanhado da palavra chave final. Caso o nome seja omitido, a classe é considerada anônima e não pode ser final.

- Classes base - lista opcional de classes pai e o modelo de herança que será usado para cada uma delas.

- Membros - lista de membros, especificadores de acesso, declaração e definição de funções de instância, tipos e classes aninhados.

2.5 Classes Abstratas

Classes abstratas são tipos que não podem ser instanciados, mas podem ser usados como classe base. Para criar uma, basta usar o especificador virtual. Classes como essas são usadas para representar conceitos gerais, por exemplo, Forma ou Animal, esses tipos serão usados como classe base para tipos concretos como Círculo ou Cachorro.

Não é possível usar tipos abstratos em parâmetros, tipos de retorno ou conversões. É possível usar esses tipos com ponteiros e referências.

2.6 Polimorfismo

Objetos de uma classe que declaram ou herdam pelo menos uma função virtual são considerados polimórficos. Dentro de cada objeto deste tipo, a implementação contém informações adicionais que são usadas por chamadas de funções virtuais e por métodos auxiliares como dynamic cast ou typeid.

2.7 Memória

C++ possui 4 tipos de gerenciamento de memória:

- Armazenamento de duração estática: são criados antes da chamada main() (salvo exceções) e destruídos na ordem reversa de criação após a saída de main().
- Armazenamento de duração de thread: Similar ao armazenamento estático, porém o objeto é criado com a thread e destruído com o join da thread.
- Armazenamento automático: Variáveis automáticas são criadas no ponto de declaração e destruídas na ordem reversa de criação de seu escopo. São alocadas automaticamente na pilha.
- Armazenamento dinâmico: São criadas com uma chamada new e destruídas com uma chamada delete.

3 IMPLEMENTAÇÃO

3.1 Orientação a Objetos

A implementação inicial inclui duas classes bases:

- Player: Classe que será utilizada para criar informações sobre o jogador, como nome e pontuação.

  Atributos: std::string name, int score.

  Métodos: Construtor, getScore, setScore(int score).

- Question: Classe que conterá uma pergunta com suas alternativas e resposta.

  Atributos: std::string enunciation, std::string alternatives[4], int answer.

  Métodos: Construtor, showQuestion(): mostra a pergunta e as alternativas, respond(int alternative): recebe a resposta do usuário, askQuestion(): realiza a pergunta com os métodos showQuestion() e respond(int alternative).

Além dessas classes, uma classe Database está incluída para gravar dados do programa, essencialmente as perguntas, e uma classe Main para execução do programa.

3.2 Recursos utilizados para a implementação OO

- Recursos mínimos sugeridos

  1. Especificar e utilizar classes (utilitárias ou para representar as estruturas de dados utilizadas pelo programa).

Listing 3.1 – Exemplo de classe utilizada no programa

```
1  #include <iostream>
2  #include <string>
3
4  class Player
5  {
6  private:
7  std::string name;
8  int score;
```

```
9       public:
10      Player(std::string name){
11      this->name = name;
12      }
13      int getScore()
14      {
15      return this->score;
16      }
17      void setScore(int score)
18      {
19      this->score = score;
20      }
21      };
```

2. Fazer uso de encapsulamento e proteção dos atributos, com os devidos métodos de manipulação (setters/getters) ou propriedades de acesso, em especial com validação dos valores (parâmetros) TODO para que estejam dentro do esperado ou gerem exceções caso contrário TODO.

Observando o Listing 3.1 o atributo **score** é protegido pelos métodos **getScore** e **setScore**, impossibilitando algum desenvolvedor de ter acesso direto à esta propriedade do objeto **Player**. Isso centraliza a leitura e a escrita desse atributo, resultando em uma melhor organização do código.

3. Especificação e uso de construtores-padrão para a inicialização dos atributos e, sempre que possível, de construtores alternativos.

O listing 3.1 também demonstra o uso de um construtor não padrão que necessita somente do atributo **name** que tem especificação de tipo, garantindo então a tipagem correta deste parametro.

4. Especificação e uso de destrutores (ou métodos de finalização), quando necessário TODO.

5. Organizar o código em espaços de nome diferenciados, conforme a função ou estrutura de cada classe ou módulo de programa TODO.

6. Usar mecanismo de herança, em especial com a especificação de pelo menos três níveis de hierarquia, sendo pelo menos um deles correspondente a uma classe

abstrata, mais genérica, a ser implementada nas classes-filhas TODO.

7. Utilizar polimorfismo por inclusão (variável ou coleção genérica manipulando entidades de classes filhas, chamando métodos ou funções específicas correspondentes) TODO.

8. Usar polimorfismo paramétrico através da especificação de *algoritmo* (método ou função genérico) utilizando o recurso oferecido pela linguagem (i.e., generics, templates ou similar) e da especificação de *estrutura de dados* genérica utilizando o recurso oferecido pela linguagem TODO.

9. Usar polimorfismo por sobrecarga (vale construtores alternativos) TODO.

10. Especificar e usar delegates TODO.

- Recursos extras

  Exemplo de recurso extra

- Recursos de processamento paralelo

  1. Definição, uso e gerência de unidades (threads, módulos, classes, métodos, funções, trechos ou instruções) de execução concorrente e o seu sincronismo TODO

  2. Definição, uso e gerência de regiões críticas (variáveis, arrays, coleções ou similares) TODO

3.3 Funcional

3.4 Recursos utilizados para a implementação Funcional

- Recursos mínimos sugeridos

  1. Priorizar o uso de elementos imutáveis e funções puras (por exemplo, sempre precisar manipular listas, criar uma nova e não modificar a original, seja por recursão ou através de funções de ordem maior). TODO

  2. Especificar e usar funções não nomeadas (ou lambda) TODO.

  3. Especificar e usar funções que usem currying TODO.

  4. Especificar funções que utilizem pattern matching ao máximo, na sua definição TODO.

5. Especificar e usar funções de ordem superior (maior) criadas pelo programador TODO.

6. Usar funções de ordem maior prontas (p.ex., map, reduce, foldr/foldl ou similares) TODO.

7. Especificar e usar funções como elementos de 1ª ordem TODO.

8.Usar recursão como mecanismo de iteração (pelo menos em funções de ordem superior que manipulem listas) TODO.

- Recursos extras

  Exemplo de recurso extra

- Recursos de processamento paralelo

  1. Definição, uso e gerência de unidades (threads, módulos, classes, métodos, funções, trechos ou instruções) de execução concorrente e o seu sincronismo TODO

  2. Definição, uso e gerência de regiões críticas (variáveis, arrays, coleções ou similares) TODO

3.5 Referências

- <https://en.cppreference.com/w/cpp/language/object>: Objects
- <https://en.cppreference.com/w/cpp/language/class>: Classes
- <https://en.cppreference.com/w/cpp/language/objectPolymorphic_objects>: Polymorphic Objects
- <https://en.cppreference.com/w/cpp/language/abstract_class>: Abstract Classes