INF224
Trabalho Prático C++/Objeto

Eric Lecolinet - **Télécom ParisTech** - **Dept. INFRES**

Links Úteis

- Página INF224 , slides do curso
- **Manuais de referência** :
  - cppreference.com
  - cplusplus. com
  - você também pode simplesmente pesquisar a classe ou função no *Google* !
- **Documentação automática com Doxygen:**
  - Manual
  - Tutorial

  E também :
  - Kit de ferramentas gráficas Qt (em C++)
  - Perguntas frequentes sobre C++
  - StackOverflow
  - Serialização com Cereal
  - Extensões de reforço

Preâmbulo

Objetivo

O objetivo deste trabalho prático é criar o esquema de software para um **set-top box multimídia** que permite reproduzir vídeos, filmes, exibir fotos, etc.

Este software será produzido por etapas, limitando-se à declaração e implementação de algumas classes e funcionalidades típicas que serão concluídas gradualmente. É útil ler o texto completo de cada etapa antes de prosseguir (especialmente quaisquer notas ou comentários que forneçam orientação).

Renderização

A renderização ocorrerá no final do UE. As duas partes (este TP) e a parte **Java**/**Swing** serão renderizadas ao mesmo tempo (porque estão vinculadas). Instruções para renderizar o laboratório podem ser encontradas no final do laboratório **Java**/**Swing** .

Será solicitado um **README** descrevendo brevemente seu trabalho, bem como um **Makefile** (veja abaixo). Lembre-se de anotar o que você fez e as respostas às perguntas à medida que avança.

Quando há modificações significativas no código fonte (em particular na função main()) é útil manter a versão anterior nos comentários ou, melhor, entre #ifdef VERSION_TRUC e #endif ,

conforme explicado no TP.

Ambiente de Desenvolvimento (IDE)

É essencial utilizar uma **ferramenta apropriada** para desenvolver código C++. Deve permitir: 1) exibir o código corretamente (com coloração de sintaxe), 2) compilar **no IDE** , 3) (se possível) depurar **no IDE** .

Alguns ambientes não oferecem os recursos 2) e 3), muitas vezes por estarem mal configurados. Se for uma escolha pessoal, faça o que achar melhor. Caso contrário, eu recomendo fortemente que você mude de IDE. As mensagens dos compiladores C++ costumam ser longas, numerosas e nem sempre claras, o que exige a capacidade de ler o código ao mesmo tempo que os erros. **Um IDE adequado economizará um tempo precioso!**

Nas máquinas da escola (ou nas suas se você instalar) você pode por exemplo usar **o QtCreator** , que tem a vantagem de ser compatível com o Makefile . É acessível na escola a partir do menu *Desenvolvimento* ou do *Terminal* digitando o comando qtcreator & . Outras ferramentas não são necessariamente configuradas para C++.

Se você quiser trabalhar em sua máquina, considere instalar o IDE com antecedência, pois esta operação pode demorar um pouco. Lembre-se também de instalar o pacote C++ (por exemplo, com Visual Studio) e prestar atenção às versões (por exemplo, Eclipse para Java vs. Eclipse para C++).

Makefile

O problema com IDEs é que eles não são **portáveis**(você deve ter o IDE em sua máquina para recompilar). **Por isso pedimos que você também crie um Makefile** . Este arquivo permitirá compilar tudo e até iniciar o programa digitando make run no Terminal. Makefile é padrão no Unix e Mac e está disponível no Windows.

Observe que um programa sem Makefile será considerado não renderizado porque não poderemos recompilar seu programa!

PT

1º Passo: Primeiros passos

1. Abra uma janela **do Terminal**
2. Crie um **diretório** para os arquivos deste TP, por exemplo: `mkdir inf224`
3. Vá para este diretório: `cd inf224`
4. Copie os arquivos **Makefile** e **main.cpp** para este diretório. **Não recorte e cole** , mas use o comando apropriado do navegador (geralmente no menu de contexto).
   - Motivo: o Makefile contém **abas** que recortam e colam às vezes se transformam em espaços!
   - Atenção: se o navegador adicionar a **extensão .txt** ao Makefile , ela deverá ser removida!
5. Está pronto, você pode iniciar o IDE e importar main.cpp .

Para criar um projeto Makefile compatível com QtCreator

- Clique em *"Novo arquivo ou projeto..."* no menu *"Arquivo"*
- Clique em *“Importar um projeto”* e depois *em “Importar um projeto existente”*

- Escolha um **nome de projeto** (o que você quiser) e o **diretório** que você criou anteriormente
- Selecione este diretório garantindo que o arquivo main.cpp esteja selecionado.
- Valide quantas vezes forem necessárias

Isso criará um arquivo que possui o nome do projeto com a extensão .creator . Este arquivo permitirá que você reabra seu projeto posteriormente clicando em *"Abrir arquivo ou projeto..."* no menu *"Arquivo"* (ou, em alguns ambientes, apenas clicando duas vezes no arquivo .creator).

2º Passo: Aula Básica

Escreva a **declaração** (arquivo .h de cabeçalho) e a **implementação** (arquivo .cpp de origem) da **classe base** da árvore de herança das classes de objetos multimídia. Esta classe base conterá o que é comum a todos os objetos de mídia. Iremos então utilizá-lo para definir subclasses específicas para cada tipo de dado (por exemplo, uma aula de foto, vídeo, filme, peça musical, etc.)

Para criar esses dois arquivos, no QtCreator clique em: *Arquivo* , depois *Novo arquivo ou projeto...* e depois *Classe C++* (o nome do .h e do .cpp será forjado a partir do nome da classe). Observe também que, por convenção, os nomes de suas classes devem começar com letra maiúscula e os de variáveis e funções com letra minúscula.

Para simplificar, esta classe base terá apenas duas variáveis de instância:

- O **nome** do objeto multimídia que deve ser do tipo **std::string** (**não** use o antediluviano char* da linguagem C!). Observe que std é o nome do **namespace** da **biblioteca padrão C++** (do qual a classe *string* faz parte).
- O **nome do arquivo** associado ao objeto de mídia. Este é o caminho completo para acessar este arquivo (uma foto jpeg, um vídeo mpeg, etc.) no sistema de arquivos Unix.

Declare e implemente dois **construtores** (um sem argumentos, outro com argumentos), o **destruidor** , bem como os **getters** e **modificadores** para poder ler ou modificar as variáveis de instância.

Declare e implemente um **método de exibição** para exibir o valor das variáveis do objeto. Este método, que será inspirado no exemplo visto em aula, não modificará o objeto e exibirá os dados em um **std::ostream** , ou seja, um **stream de saída** . Isso permitirá que você use a mesma função para exibir no Terminal, em um arquivo ou em um buffer de texto, o que será útil posteriormente. Concretamente:

- sua função deve ter um parâmetro do tipo std::ostream & . Não se esqueça do & veremos para que serve mais tarde.
- para exibir no Terminal, esta função precisará apenas ter std::cout (""console out") como argumento, ou seja, a **saída padrão** .
- seu método de exibição **não** deve criar ou retornar uma string como *toString()* faria em Java (faremos isso mais tarde usando std::stringstream).

Além disso, pense em:

- inclua em seu cabeçalho as declarações dos objetos da biblioteca padrão que você precisa, ou seja, pelo menos os cabeçalhos string e iostream (esses cabeçalhos declaram as classes std::string e std::iostream).
- use a palavra-chave const para funções onde desejar (**todas** as funções que não modificam variáveis de instância devem ser const). Esta palavra-chave é usada para evitar erros (ou seja, para evitar a modificação de variáveis que não deveriam ser modificadas inadvertidamente).

Para compilar o arquivo e corrigir erros

- Abra o arquivo Makefile para modificar:
  - PROG : o nome do programa que deseja produzir
  - SOURCES : a lista de seus arquivos **.cpp** separados por espaços
  Não coloque o .h em SOURCES
- Se você usa QtCreator :
  - você eventualmente terá que clicar em *Lista de Projetos* na barra superior e selecionar *Arquivos de Sistema* para abrir os arquivos no diretório de trabalho (em particular o arquivo Makefile)
  - Clicar no martelo (canto inferior esquerdo) ou Control-B inicia a compilação. Uma barra horizontal indica então o estado da compilação: enquanto estiver verde está tudo bem, se ficar vermelho há um erro! Neste caso, clique na barra para exibir a lista de erros. **Clicar em um erro exibe a linha que causou o erro** .
- Em todos os casos ocorrerá um erro **de edição do link** se o arquivo main.cpp não existir ou se você não o tiver adicionado ao SOURCES do *Makefile* (ver próxima pergunta).

3ª Etapa: Programa de teste

Um **programa executável** requer uma função main() . Esta função não deve ser encontrada na implementação de uma classe porque isso proibiria sua reutilização. Portanto, iremos implementá-lo em outro arquivo, aqui main.cpp .

Para testar, crie algumas instâncias da classe base (usando new) em main() e verifique se a função display exibe corretamente o valor dos atributos no Terminal. Observe que seu código deve (e deve posteriormente) respeitar o **princípio do encapsulamento : variáveis de objeto nunca devem ser acessadas de outra forma que não seja através de métodos.**

- **Para compilar:** faça como antes ou digite o comando make no Terminal no diretório do projeto.
- **Para executar o programa:** digite make run no Terminal ou digite o nome do programa precedido de ./ (exemplo: ./myprog)

Pensar:

- A **Recue** seu código-fonte: com a maioria dos IDEs, Control-I (ou TAB) **recua automaticamente** a linha atual e Control-A Control-I recua todo o texto.
- Para tornar seu programa o mais **legível** possível:
  - separando **os** métodos com linhas brancas,
  - colocando espaços **ao redor de =** e **depois de , e de ;**
  - empacotamento para que seu código não tenha mais do que 80 **caracteres** de largura. As quebras de linha não afetam a compilação, mas tornam o código mais legível (o que reduz erros)!
- Para **comentar** , em particular nos cabeçalhos para poder gerar automaticamente a documentação usando **Doxygen** . Podemos nos limitar às funcionalidades básicas do Doxygen.

Etapa 4: fotos e vídeos

Criaremos agora duas **subclasses** da classe base, uma correspondente a uma **foto** e outra a um **vídeo** . Essas classes podem incluir muitos atributos, mas vamos mantê-las simples para não perder tempo:

- um **vídeo** tem uma **duração** , ou seja, um valor numérico integral,
- uma **foto** pode ser caracterizada por uma **latitude e uma longitude** , ou seja, dois valores numéricos reais.

Estas duas classes deverão ser declaradas em arquivos próprios para obter maior modularidade e facilitar a reutilização. Sendo essas classes simples, podemos **implementá-las nos cabeçalhos** (nesse caso não deveria haver um arquivo .cpp).

Declare e implemente construtores, acessadores, modificadores e o método display (que deve ter a mesma assinatura da classe pai, incluindo const). Não se esqueça de inicializar **variáveis que possuem tipos base** em construtores, caso contrário seu valor será **aleatório** (ao contrário dos objetos, que são inicializados automaticamente graças aos seus construtores).

Por fim, declare e implemente um método que permita **reproduzir o objeto multimídia** , ou seja, dependendo do caso, exibir a foto ou reproduzir o vídeo. Concretamente, esta função chamará outro programa (por exemplo, no *Linux* , "mpv" para um vídeo ou "imagej" para uma foto) através da função system() padrão , exemplo:

```
sistema("caminho mpv &"); //caminho é o caminho completo
```

Para criar o argumento system() , simplesmente concatene a **string** s com o operador + e, em seguida, chame o método data() da string resultante para convertê-la em char * (porque system() recebe um argumento char *). Não se esqueça de & para iniciar o outro programa em segundo plano.

Quanto à função de exibição:

- a função para reproduzir o objeto não deve modificar o objeto
- deve ser declarado nas classes **Photo** e **Video** mas também na **classe base** para permitir **polimorfismo**

Ao contrário da função de exibição:

- ele não pode ter uma implementação no nível da classe base (porque cada tipo de objeto requer um programa diferente para ser executado).

Como é chamado esse tipo de método e como devem ser declarados?

Modifique o Makefile se necessário (lembre-se que você não deve colocar .h em SOURCES). Compile, corrija quaisquer erros e teste o programa.

Caso você tenha feito o acima solicitado, não será mais possível instanciar objetos da classe base. *Para que ?*

Etapa 5: Processamento uniforme (usando polimorfismo)

Agora queremos ser capazes de processar uniformemente **uma** lista incluindo fotos e vídeos sem precisar nos preocupar **com seu tipo** .

Para fazer isso, crie uma **tabela em** main.cpp cujos elementos ora são uma foto e ora um vídeo. Em seguida, escreva um loop que permita exibir os atributos de todos os elementos do array ou “reproduzi-los”. Este loop deve tratar todos os objetos derivados da classe base da mesma maneira.

- Qual é a propriedade característica da **orientação a objetos** que permite que isso seja feito?
- O que é especificamente necessário fazer **no caso de C++** ?
- Qual é o **tipo dos elementos do array** : o array deve conter objetos **ou** ponteiros **para** esses objetos? Para que ? Compare com Java.

Compile, execute e verifique se o resultado está correto.

6º passo. Filmes e pinturas

Queremos agora definir uma subclasse **Film** derivada da classe **Video** . A principal diferença é que os Filmes terão **capítulos** que permitem acessar rapidamente parte do filme. Para fazer isso usaremos um **array** de inteiros contendo a **duração** de cada capítulo.

Escreva a classe **Film** , que deve ter:

- o (s) **fabricante** (s) apropriado(s) .
- um **modificador** que permite passar uma **série de durações** como argumento (leia atentamente as notas abaixo)
- um (ou mais) **acessador(es)** retornando a matriz de durações e o número de capítulos
- um **método de exibição** mostrando a duração dos capítulos (o método para reproduzir o objeto não precisa ser redefinido)

Notas:

- Um programador experiente usaria um std::vector (que veremos mais tarde) em vez de um array. Este não é o caso aqui porque o objetivo desta questão é mostrar algumas dificuldades.
- Em C/C++, um array é referenciado por um **ponteiro** . Quando o número de elementos muda, você deve criar um novo array e destruir o anterior. Quando não há elemento, um array não é criado e o ponteiro deve ser nullptr .
- Em C/C++, quando você passa um array como argumento para uma função, seu **endereço de memória** é copiado para o parâmetro da função (que é na verdade um ponteiro). Isto tem duas consequências:
  - A função não pode **saber o número de elementos** no array, a menos que **também o passe como argumento** (ou use uma convenção, por exemplo char * em C sempre termina com 0). É diferente em Java (arrays Java sabem o número de elementos).
  - Os elementos **não são copiados** (apenas o endereço do array é copiado).

Sabendo do que foi dito acima, a questão é como o **modificador** deve **armazenar o array** no objeto Movie. Considere os seguintes pontos:

- O array fornecido como argumento pode ser **destruído** ou **reutilizado** no futuro
- Por outro lado, o objeto Filme deve garantir a **longevidade dos seus dados** . Se seus dados se tornassem subitamente inválidos, não apenas o **princípio do encapsulamento** seria quebrado (o objeto perderia o controle de seus dados), mas o programa teria uma boa chance de **travar** !

Da mesma forma, pense no que o **acessador** deve retornar. na verdade, não deveria permitir que o chamador modificasse o conteúdo da matriz (o que também quebraria o **princípio do encapsulamento**).

Implemente sua classe e verifique se o resultado está correto modificando e/ou destruindo o array passado como argumento e depois chamando a função de exibição do objeto (NB: você deve repetir essas operações para que a memória seja reaproveitada).

Nota: essas questões não são específicas de arrays, elas surgem sempre que um modificador tem um ponteiro ou referência como argumento. A questão realmente é **quem possui e controla os dados** .

7º passo. Destruição e cópia de objetos

Ao contrário do Java, o C/C++ não gerencia **a memória dinâmica automaticamente** : como não há **coleta de lixo** , o que é criado com new ocupa memória até o programa terminar, a menos que seja destruído com delete . (Nota: este problema ocorre apenas com o que é criado com new , delete nunca deve ser usado em nenhum outro caso).

Dentre as classes escritas anteriormente, quais devem ser modificadas para que não haja **vazamento de memória** quando destruirmos os objetos? Altere o código para evitar isso.

A cópia de objetos também pode ser um problema quando eles possuem variáveis de instância que são ponteiros. Qual é o problema e quais são as soluções? Implemente um.

8º passo. Criar grupos

Criaremos agora uma nova classe usada para conter um **grupo** de objetos derivados da classe base. Um grupo pode conter um conjunto de objetos semelhantes (por exemplo, um grupo para todas as fotos e outro para todos os vídeos) ou não (por exemplo, um grupo para fotos e vídeos de férias).

Para fazer isso usaremos a **classe template** std::list< > da biblioteca padrão que permite criar uma lista de objetos (cuja classe deve ser especificada entre o < >). Observe que se trata de usar uma classe de modelo existente, não de criar uma nova.

Duas estratégias são possíveis:

- crie uma classe de grupo que **contenha** uma lista de objetos (a lista é então uma variável de instância da classe)
- crie uma classe de grupo que **herda** uma lista de objetos (a lista é então herdada da classe pai)

A primeira estratégia requer a definição de métodos na classe de grupo para gerenciar a lista. A segunda estratégia evita esse trabalho porque permite herdar métodos de std::list . Portanto, é mais rápido de implementar, mas oferece menos controle (não escolhemos os nomes dos métodos como queremos e herdamos todos os métodos de std::list incluindo alguns que talvez sejam inúteis ou indesejáveis).

Para ir mais rápido, escreva esta classe usando a segunda estratégia. Defina os seguintes métodos:

- Um **construtor**
- um **acessador** que retorna o nome do grupo
- um **método de exibição** que exibe os atributos de todos os objetos na lista

O grupo não deve destruir objetos quando for destruído porque um objeto pode pertencer a vários grupos (veremos este ponto na próxima questão). Lembramos também que a lista de objetos deve ser na verdade uma lista de **ponteiros** de objetos . Para que ? Compare com Java.

Para testar, crie alguns grupos em main() preenchendo-os com fotos, vídeos ou filmes e fazendo com que os objetos **pertençam a vários grupos** . Chame a função de exibição de grupo para verificar se está tudo bem.

9º passo. Gerenciamento automático de memória

Como vimos nas etapas 6 e 7, o gerenciamento da memória dinâmica (alocada com new em C++ e malloc() em C) é delicado. Corremos o risco de acabar com **ponteiros pendentes** porque o objeto para o qual eles apontaram foi destruído em outro lugar (veja o passo 6), ou com **vazamentos de memória** porque não destruímos objetos que não estão mais apontados para lugar nenhum (veja o passo 7).

Les **pointeurs pendants** sont une source majeure de **plantages** ! Les **fuites mémoires** posent surtout problème si les objets sont gros (e.g. une image 1000x1000) et/ou si le programme s'exécute longtemps (e.g. un serveur Web tournant en permanence). On peut alors rapidement épuiser toute la mémoire disponible (noter cependant que la mémoire allouée de manière standard est toujours récupérée à la terminaison du programme).

Le ramasse miettes de Java et les les **smart pointers avec comptage de références** de C++ offrent une solution simple à ce problème : les objets sont alors **automatiquement détruits** quand plus aucun (smart pointer) ne pointe sur eux. Il ne faut donc jamais detruire avec delete un objet pointé par un smart pointer !

Le but de cette question est de remplacer les **raw pointers** (les pointeurs de base du C/C++) par des **smart pointers non intrusifs** dans les groupes de la question précédente. Les objets seront alors automatiquement détruits quand ils ne seront plus contenus par aucun groupe. Pour ce faire, utilisez les shared_ptr< > qui sont standard en C++11. Afin de vérifier que les objets sont effectivement détruits, modifiez leurs destructeurs de telle sorte qu'ils affichent un message sur le Terminal avant de "décéder".

Remarques

- Cette question ne porte pas sur les Films, qu'il n'est pas souhaitable de modifier.
- Pour simplifier l'expression, vous pouvez créer de nouveaux types grâce à using ou typedef en les déclarant dans le ou les headers appropriés:

```
using TrucPtr = std::shared_ptr<Truc>;
typedef std::shared_ptr<Truc> TrucPtr;
```

Enlevez des objets des groupes et vérifiez qu'ils sont effectivement détruits quand ils n'appartiennent plus à aucun groupe (et s'ils ne sont plus pointés par aucun autre smart pointer : noter que si `p` est un smart pointer `p.reset()` fait en sorte qu'il ne pointe plus sur rien)

10e étape. Gestion cohérente des données

On va maintenant créer une classe qui servira à fabriquer et manipuler tous les objets de manière cohérente. Elle contiendra deux variables d'instance:

- une table de tous les objets multimédia
- une table de tous les groupes

Para encontrar elementos de forma eficiente com base em seus nomes, essas tabelas não serão tabelas ou listas, mas **tabelas associativas (consulte o modelo map class)**. Cada elemento estará associado a uma **chave** do tipo string que será, dependendo do caso, o nome do objeto ou do grupo. Primeiro comece escrevendo esta classe, como de costume, em um novo arquivo. Como antes, usaremos ponteiros inteligentes para gerenciar a memória automaticamente.

Declare e implemente métodos apropriados para:

- **Crie** uma foto, um vídeo, um filme, um grupo e adicione-os à tabela apropriada. Observe que você precisa de um método para cada tipo, retornando o objeto criado, para poder manipulá-lo posteriormente (ou seja, chamar as funções declaradas para este tipo)
- **Pesquise** e **exiba** um objeto multimídia ou grupo com base em seu nome, fornecido como argumento. O objetivo será exibir os atributos de um objeto ou o conteúdo de um grupo no Terminal (se existir, caso contrário exibir uma mensagem de alerta).
- **Reproduza** um objeto multimídia (a partir de seu nome, dado como argumento). A mesma coisa de antes, exceto que chamamos o método play() em vez de exibir os atributos.

Para implementar esses métodos você provavelmente precisará dos seguintes métodos: map::operator[] (para insert), map::find () e map::erase (). Em seguida, faça alguns testes em main.cpp para verificar se esses métodos funcionam corretamente.

Os métodos anteriores garantem a **consistência do banco de dados** porque quando criamos um objeto o adicionamos à tabela apropriada. Por outro lado, este não será o caso se criarmos um objeto diretamente com new (ele não pertencerá a nenhuma tabela). Como podemos proibi-lo, para que apenas a classe usada para manipular objetos possa criar novos?

Pergunta adicional (você pode pular esta pergunta se estiver atrasado):

- Adicione métodos para **excluir** um objeto ou grupo de mídia com base em seu nome, fornecido como argumento. O elemento deve ser removido da tabela apropriada e destruído se não pertencer mais a nenhuma tabela. Além disso, ao excluir um objeto multimídia você deve primeiro removê-lo de todos os grupos em que aparece.

11º passo. Servidor cliente

Cette étape vise à **transformer votre programme C++ en un serveur** qui communiquera avec un client qui fera office de télécommande. Dans cette question le client permettra d'envoyer des commandes textuelles. Plus tard, dans le TP suivant, vous réalisez une interface graphique **Java Swing** qui interagira avec le serveur de la même manière. Dans la réalité le serveur tournerait sur la set-top box et le client sur un smartphone ou une tablette.

Récuperez **ces fichiers**, qui comprennent un client et un serveur ainsi que des utilitaires qui facilitent la gestion des sockets (le client est constitué des fichiers *client.cpp, ccsocket.cpp* ; le serveur des fichiers *server.cpp, tcpserver.cpp, ccsocket.cpp*). Pour les compiler vous pouvez taper : make -f Makefile-cliserv dans le Terminal.

Le **serveur** doit être lancé en premier et terminé en dernier.

- Le **client** crée une Socket qu'il connecte au serveur via la méthode connect(). Celle-ci précise **à quelle machine et à quel port** il faut se connecter. Par defaut le port est **3331** (le même que pour le serveur) et la machine est **127.0.0.1**, ce qui signifie que le client doit tourner sur la même machine que le serveur. La méthode connect() renvoie 0 si la connexion réussit et une valeur négative en cas d'erreur.

  S'il n'y a pas de firewall bloquant les connexions le **client** peut tourner sur une autre machine à condition de mettre l'adresse de la machine du serveur à la place de 127.0.0.1.

  Si la connexion est réalisée, le client lance une boucle infinie (pour quitter, taper ^C ou ^D) qui demande une chaîne de cacatères à l'utilisateur puis l'envoie au serveur via la méthode writeLine(). Le client **bloque** jusqu'à la réception de la réponse retournée par le serveur qui est lue par la méthode readLine().

- Côté **serveur**, une lambda est donnée en argument à TCPServer(), le constructeur du serveur. Cette lambda sera appelée chaque fois que le serveur recevra une requête du client. La lambda recupère la requête via son argument request, effectue un traitement, puis retourne une réponse vers le client via son argument response.

  Por enquanto, esta função apenas copia "OK:" seguido do valor de request em response . **É esta função que você precisará adaptar** para que ela faça o processamento desejado.

  Nota: lembre-se que lambdas podem **capturar as variáveis**da função onde são chamadas (e o ponteiro this oculto dos objetos).

O **cliente** e **o servidor** utilizam as classes Socket , ServerSocket e SocketBuffer que facilitam o uso de soquetes:

- Socket é usado para criar um soquete no modo **TCP**
- ServerSocket é um soquete especial que permite a um servidor detectar solicitações de conexão de clientes
- SocketBuffer é usado para simplificar o tratamento de mensagens. No caso geral, os soquetes TCP **não preservam os limites entre as mensagens** , portanto uma mensagem pode chegar em várias partes ou pelo contrário ficar presa à mensagem anterior. Os métodos SocketBuffer writeLine() e readLine() resolvem esse problema

adicionando um delimitador no final de cada mensagem. Por padrão é o caractere '\n' , **portanto suas mensagens não devem contê-lo.**

Observe que :

- O cliente e o servidor usam o protocolo conectado TCP/IP que permite a troca de um fluxo de bytes entre dois programas. Este protocolo garante que não haja pacotes perdidos e que eles cheguem sempre em ordem.
- A conexão é persistente : quando você inicia o cliente, ele se conecta ao servidor e permanece conectado até que o cliente **termine .** Outra possibilidade seria estabelecer uma nova conexão e fechá-la a cada troca cliente/servidor (é o que um navegador web faz por padrão).
- O diálogo é síncrono : o cliente envia uma mensagem e **bloqueia** até receber a resposta. Isso simplifica a programação, mas não é o ideal porque o cliente irá travar se o servidor não responder (ou se demorar muito para responder). Isto é particularmente irritante quando o cliente é uma GUI.
- Le serveur utilise des threads et peut être connecté à plusieurs clients simultanément. Ceci poserait des problèmes de concurrence si plusieurs clients modifiaient les données du serveur en même temps (mais nous n'aurons qu'un seul client).

A vous de jouer :

En vous inspirant de server.cpp, adaptez votre propre programme pour qu'il joue le rôle du **serveur de la set-top box** et appelle les fonctions adéquates **chaque fois qu'il recoit une requête du client**. Pour ce faire il faudra définir un **protocole de communication** simple entre votre serveur et le client (client.cpp que l'on remplacera plus tard par une interface Java Swing). Il faut au moins pouvoir:

- **rechercher** un objet multimédia ou un groupe et **afficher ses attributs** sur la "télécommande" (c'est-à-dire, pour l'instant, les envoyer au client qui les affichera sur le Terminal)
- **jouer** un objet multimédia (l'objet doit être joué sur la "set-top box", c'est-à-dire le serveur).

Remarques

- Afin de conserver la version précédente du code dans le main() vous pouvez le mettre entre `#ifdef VERSION_TRUC` et `#endif` (il y a aussi `#if` et `#else` et il suffit d'écrire `#define VERSION_TRUC` en haut du fichier pour que ce soit cette version qui soit compilée).
- Pensez à ajouter les fichiers *ccsocket.cpp* et *tcperver.cpp* à votre projet. Sous Linux modifier la macro LDLIBS= de telle sorte que l'on ait LDLIBS= -lpthread dans le Makefile.
- Vous aurez besoin de découper la requête en plusieurs mots. Pour ce faire vous pouvez utiliser les méthodes de std::string ou un **std::stringstream**. Un stringstream est un "buffer de texte" sur lequel on peut appliquer l'operateur >> ou la fonction **getline()** comme avec l'entrée standard std::cin ou les fichiers. La fonction getline() permet de découper jusqu'à la fin de ligne ou un caractère qui lui est donné en argument. L'operateur >> s'arrête dès qu'il trouve un espace.
- Une solution simple pour renvoyer les attributs d'un objet au client est d'utiliser sa méthode d'affichage en lui passant en argument un (autre) stringstream au lieu de std::cout. Le stringstream peut ensuite être converti en std::string via sa méthode str().
- **Attention**, comme dit précédemment, il ne faut pas envoyer les caractères '\n' ou '\r' au client ni au serveur (donc enlever les std::endl) car ils servent à délimiter les messages qu'ils échangent. Ces caractères sont également utilisés comme délimiteurs par Java, ce qui sera utile dans la suite du TP.

Questions additionnelles (vous pouvez passer ces questions si vous êtes en retard) :

- Vous pouvez rajouter **d'autres commandes**, par exemple chercher tous les objets commencant par ou contenant une séquence de caractères, ou étant d'un certain type, ou

encore afficher toute la base ou détruire un objet ou un groupe.

- Un véritable serveur aurait probablement de nombreuses commandes. Afin d'accélérer la recherche des commandes vous pouvez utiliser une std::map contenant des **pointeurs de méthodes** ou des **lambdas** (la clé est le nom de la commande, l'attribut est la méthode ou la lambda correspondante).

12e étape. Sérialisation / désérialisation

C++ ne propose pas en standard de moyen de **sérialiser** les objets de manière portable. On peut utiliser des extensions pour le faire (par exemple **Cereal**, Boost ou le toolkit **Qt**) ou juste l'implémenter "à la main" dans les cas simples. C'est ce que l'on va faire maintenant pour les tables d'objets multimédia. Notez qu'il est avantageux d'implémenter en même temps les fonctions d'écriture et de lecture, ces deux fonctionnalités dependant l'une de l'autre.

Ecrivez toutes les méthodes nécessaires en vous inspirant du cours puis testez les dans main() en sauvegardant puis en relisant la table d'objets multimédia (on laisse de côté les groupes). On rappelle:

- qu'il faut utiliser **std::ofstream** pour écrire sur un fichier, **std::ifstream** pour lire depuis un fichier et qu'il est nécessaire de vérifier si les fichiers ont pu être ouverts et de les fermer après usage à l'aide de leur méthode close().
- qu'il faut écrire les attributs des objets de telle sorte que l'on puisse ensuite les lire de manière non ambiguë. Faire en particulier attention aux chaînes de caractères, qui peuvent contenir des espaces. La fonction **getline()** résout ce problème.
- que pour lire - et donc créer - un objet multimédia à partir d'un fichier, il faut connaître sa classe. Il faut donc enregistrer le nom de la classe lors de l'écriture. Une solution est de définir pour chaque classe une méthode retournant son nom (sinon voir remarques plus bas).
- qu'il est souhaitable d'implémenter une **fabrique d'objets** (factory) permettant de créer les objets à partir du nom de leur classe.
- enfin, bien sûr il ne faut pas oublier les principes de l'orienté objet : chaque objet est le mieux placé pour savoir comment il doit être lu ou écrit sur un fichier et l'implémentation doit se faire en conséquence

Questions additionnelles

- Faire en sorte que l'on puisse utiliser les opérateurs operator<< et operator>> pour sérialiser et désérialiser les objets. Comme ils ne peuvent pas être déclarés comme des méthodes virtuelles des classes, il faut seulement les définir pour la classe de base et appeller une méthode virtuelle (redéfinie pour chaque sous-classe) pour faire le travail.
- Faites en sorte de pouvoir également (dé)sérialiser la table des groupes, en plus de la table des objets. Attention : ne pas dupliquer les objets (qui peuvent appartenir à plusieurs groupes !). Alternativement, vous pouvez utiliser une extension comme **Cereal**.

Remarque: C++ fournit un moyen de récupérer les noms des classes via la fonction typeid(), mais leur format dépend de l'implémentation et n'est donc pas portable (par exemple *N7Contact8Address2E* pour *Contact::Address* avec g++) La solution la plus simple est donc de faire comme conseillé plus haut. Il existe aussi des extensions pour décoder les noms, par exemple avec g++ :

```
#include <cxxabi.h>

bool demangle(const char* mangledName, std::string& realName) {
    estado interno = 0;
    char* s = abi::__cxa_demangle(mangledName, NULL, 0, &status);
    nomereal = (status == 0) ? s: nome mutilado;
    grátis(s);
    status de retorno;
}
```

```
std::string nome real;
demangle(typeid(Foto).nome(), nome real);
cout << nome real << endl;

ptr<Foto> p;
demangle(typeid().name(), nome real);
cout << nome real << endl;
```

13º passo. Manipulação de erros

A confiabilidade dos programas depende da qualidade do tratamento de erros durante a execução. Na verdade, é necessário evitar a produção de resultados inconsistentes ou falhas resultantes de manipulações incorretas. Até agora negligenciamos esse aspecto nas questões anteriores.

Exemplos:

- se criarmos vários grupos ou objetos com o mesmo nome
- se excluirmos um grupo ou objeto que não existe
- se o nome contiver caracteres inválidos
- se a tabela de duração de um filme tiver tamanho zero ou menor que zero

Existem basicamente duas estratégias para lidar com erros. A primeira é retornar **códigos de erro** (ou um ponteiro booleano ou nulo) ao chamar uma função que pode produzir erros. A segunda é gerar exceções .

No primeiro caso (códigos de erro), é desejável que a função execute uma ação "razoável" em caso de erro (por exemplo, não faça nada em vez de travar se pedir para excluir um objeto que não existe!). Esta solução pode representar dois problemas:

- a propagação de erros entre funções nem sempre é fácil de tratar corretamente (o erro pode ocorrer em uma função chamada em uma função chamada em outra função, e assim por diante).
- os programadores, sempre apressados, tendem a negligenciar os códigos retornados. Isto pode tornar o comportamento de programas complexos imprevisível e complicar consideravelmente a depuração.

A segunda solução (exceções) é mais segura na medida em que os erros devem ser tratados através de uma cláusula catch , caso contrário o programa será encerrado (ou será proibido de compilar em Java). No entanto, o uso excessivo de exceções pode complicar o código e dificultar a compreensão de seu fluxo.

Você decide ! Trate os principais casos de erro conforme achar adequado, usando a primeira ou a segunda estratégia, ou uma combinação das duas, dependendo da gravidade dos erros. Mas certifique-se de que seu código esteja consistente com suas escolhas e justifique-as no relatório e/ou na documentação gerada pelo **Doxygen** .

Nota: para criar novas classes de exceção em C++ é preferível (mas não obrigatório) subclassificar uma classe existente da biblioteca padrão. A exceção **runtime_error** (que deriva da classe **de exceção**) é particularmente apropriada e seu construtor recebe como argumento uma mensagem de erro do tipo string . Esta mensagem pode ser recuperada quando a exceção é capturada usando o método what() (veja o exemplo no final desta **página**).

Seguindo

O resto deste TP (controle remoto em Java Swing) está **disponível aqui** .

Eric Lecolinet - http://www.telecom-paristech.fr/~elc - Telecom ParisTech