VF 2018

1) **Explique a diferença entre objeto e classe, citando os modificadores de acesso que podem ser atribuídos aos seus membros e descrevendo suas funções.**
   Classe = é o modelo dos objetos, dizendo quais características pode ter, servindo como um esqueleto para a construção do objeto.
   Objeto = algo concreto, com especificações nas características, podendo ser manipulado. É a materialização da classe
   Modificadores de acesso:
   Private= atributos ou métodos não podem ser chamados de fora da classe, nem ser herdados por objetos de classes distintas
   Protected= atributos ou métodos não podem ser chamados de fora da classe, mas pode ser herdados por objetos de classes distintas
   Public= atributos ou métodos podem ser chamados de fora da classe e ainda ser herdados por objetos de classes distintas

2) **Explicar porque o código mostrado apresenta erro de compilação e resolver esse erro mexendo nas classes (sem modificar a main).**

   1. A classe triangle na hora de ser criada por herança não coloca públic do lado que nem a rectangle
   2. Dessa forma o método set_value é herdado como private(quando não se coloca nada , é private ...default) da classe polygon . Isso quer dizer que ele não pode ser chamado de fora da classe . Dessa forma ppoly2->set_value(4,5) tem que ser tirado pq ppoly2 aponta para um Triangle cujo método set_value é private
   3. A classe triangle tem dois atributos em seu construtor. Eles devem ser colocados na hora de se criar o objeto
   Triangle trgl(4,5). Caso contrário vai dar erro.

3) **Explicar diferença entre linguagem estruturada e a orientada em objeto (igual a 6 antes do glossário git lá embaixo)**

4) **Sei lá**

5) **O que são classes abstratas e quais suas funções.**
   São classes que servem como rascunho para ditar o comportamento de classes herdadas delas. São usadas como modelos para criação de outras classes.
   Com o polimorfismo, elas podem receber comportamento através das classes concretas (derivadas).

VC 2018

1) **O que são bibliotecas, bibliotecas estáticas e dinâmicas e quais suas vantagens e desvantagens?**

Uma biblioteca é uma coleção de recursos pré-compilados utilizados por programas de computador, podendo incluir dados de configuração, documentação, dados de ajuda, modelos de mensagem, código pré-escrito e sub-rotinas, classes, valores ou especificações de tipo.

Bibliotecas são coleções de funções pré-compiladas, relacionadas na Gui Library, dbm Library ou mesmo criada pelo programador, com facilidade de aplicação que mantém a exibição do código preservada. Sua utilização serve para reusar funções e reduzir o número de linhas do código do arquivo-fonte.

Biblioteca estática: É um conjunto de rotinas, funções e variáveis inseridas no programa executável no momento da compilação, especificamente na linkagem, quando há a combinação do código com a biblioteca estática.

Vantagem: Ter certeza de que todas as bibliotecas estão presentes no programa e na versão correta, evitando problemas de dependência.

Desvantagem: O tamanho do executável é maior que na biblioteca dinâmica

A biblioteca é descartada após seu uso.

Biblioteca dinâmica: É um conjunto de rotinas, funções e variáveis que ficam carregados em apenas uma instância de memória quando na execução do programa.

Vantagem: Executáveis de menor tamanho

Evita duplicação da biblioteca, múltiplos programas da mesma biblioteca e apenas precisa-se de uma cópia para o sistema.

Desvantagem: A biblioteca deve estar disponível no sistema durante a execução do programa. Se não achar a biblioteca, ele não roda.

4) Classifique as linguagens BASIC 64, C e Python com relação aos paradigmas de programação, justificando sua resposta.

- Basic 64 é uma linguagem não estruturada, pois é constituída por uma série de comandos ordenados e identificados por linhas. É uma linguagem também imperativa, pois define a sequência das operações baseadas em sentenças que mudam o estado do programa.
- C: É um exemplo de linguagem procedural, pois deve-se informar ao computador cada passo a ser executado, há a existência de rotinas e sub-rotinas ou funções para agrupamento do código. A programação procedural é, de modo geral, imperativa. É também uma linguagem estruturada, uma vez que se utiliza de

diferentes estruturas para controle do fluxo do programa e há a reutilização dos subprogramas através de bibliotecas sob a forma de arquivos de cabeçalho, por exemplo (.h). Compilada: Compila e depois executa.

- Python: É uma linguagem de programação orientada a objeto, pois agrupa o código junto com os dados que o código modifica. É uma linguagem estruturada, pois se utiliza de diferentes estruturas para controle do fluxo de programa. Também pode ser considerada uma linguagem procedural e imperativa. Interpretada: Executa e compila paralelamente.

5) Descreva as etapas do processo de compilação citando suas entradas e saídas. (igual a 1 abaixo)

1) Descreva as etapas de compilação (cabeçalho):

Cabeçalho é o ínicio do código fonte, com as informações iniciais (principais).

Compilar é transformar um código em linguagem de programação para um código que a máquina entenda(executável).

As etapas são: Pré-Processamento, Compilação, Montagem e Ligação.

1. Pré Processamento: Pega o arquivo .c com o cabeçalho .h e realiza algumas ações: substituição de texto (#define, por exemplo), exclusão de comentários e inclusão de arquivos (com o #include). Acaba sendo gerado um arquivo .i com todas essas ações já feitas no programa original. Uma versão “limpa” do programa. (.c -> .i)
2. Compilação: A compilação se resume a pegar o arquivo vindo do Pré-Processamento e traduzí-lo da linguagem de programação para a linguagem assembly. (.i -> .s)
3. Montagem: Neste processo é feita a tradução do código da linguagem assembly para o código de máquina (criação do programa objeto). (.s -> .o)
4. Linkagem/Ligação: Caso haja referências a outras bibliotecas, este processo destina-se a junção desses códigos separados em um só programa executável. (.o -> executável)

Coisas q ele falou em sala: Pq o arquivo .o tem menos texto que o executável?? Pq depois que passa pela linkagem, várias bibliotecas são adicionadas ao código, aumentando seu texto.

Linkagem dinâmica: Só puxa pra memória a biblioteca que está sendo usada no momento da execução do código.

Linkagem estática: puxa do HD todas as bibliotecas necessárias

2)Entregar um código Git e Github;

1. Botão direito na pasta: gitbash here
2. git config --global user.name “nome”
3. git config --global user.email “email”
4. Criação da pasta: mkdir NomedaPasta
5. Entrar na pasta: cd NomedaPasta

6. Baixar o repositório: git clone URL do repositório
7. Sair da Branch master: git checkout nomedaoutrabranch
8. comando pull: atualiza a pasta
9. git status (compara com o original e mostra arquivos separados)
10. git add nomedoarquivo (adiciona na branch)
11. git add remote origin master URL (liga repositório local com o remoto)
12. git commit -a -m "Mensagem do commit" (permite o envio pro repositório)
13. git reset --soft HEAD~; (desfaz commit mantendo arquivos staged)
14. git reset HEAD~; (desfaz commit tornando arquivos unstaged)
15. git reset --hard HEAD~; (desfaz arquivo e remove todas as alterações)
16. git push (envia pro repositório)

3)Processo de computação. Exemplo: como o computador se inicia.

O computador se resume a 3 componentes: processador, memória (RAM e HD) e os dispositivos de entrada e saída (teclado, mouse, monitor).

Processador realiza as instruções.

Memória:

1. HD = faz o armazenamento de todos as informações que não estão sendo usadas.
2. RAM = é nela que estão as informações que estão sendo processadas, após serem usadas, elas voltam ao HD. São mais rápidos que o HD, porém possuem menor tamanho.

BOOT do computador:

1. Placa mãe ligada.
2. Um sistema chamado BIOS, gravado no chipset é ativado.
3. Contato do BIOS com a memória RAM, a placa de vídeo e outros dispositivos (reconhecimento dos aparelhos).
4. Teste realizado pelo BIOS para verificar o funcionamento da máquina.
5. Procura pelo sistema operacional.
6. Leitura do HD para achar o código que faz a inicialização do sistema operacional.
7. Inicialização do KERNEL, que serve para fazer a comunicação entre software e hardware, carregando os arquivos principais e informações básicas do sistema.
8. Início do Windows.

4)Classifique e explique: os termos. Dê exemplos. (paradigmas de computação)

Paradigmas de programação são formas de classificar as linguagens de acordo com suas características e funções.

- Imperativa = determina a ordem que as operações serão feitas. Usa-se algoritmos por meio das sentenças (a=1; some a+a).
- Procedural = determina grupos de ordens, funções.
- Declarativa = se preocupa apenas em determinar a existência e a condição para que tal ordem seja efetuada.
- Não estruturada = sequência de comandos ordenados. Usado nas primeiras linguagens de programação. Eleva o desempenho e são

mais próximos da linguagem da máquina, porém podem ser difíceis de ler. Ex: BASIC

- Estruturada = criada para padronizar e melhorar a clareza, qualidade. Pode ser por blocos ou orientada a objeto:
    1. Por blocos: definição do escopo do código por delimitadores (na linguagem C seriam as chaves)
    2. OO: vai agrupando os códigos de acordo com os dados que se modificam.

C é estruturada, procedural, imperativa.

5)Explique como a aproximação da linguagem aproximada dos objetos reais poderia melhorar a programação?

?????????????/

Teríamos mais bagagem e entendimento de como funcionam as classes na linguagem orientada a objeto, podendo repartir um problema em partes, entendendo essas partes de uma forma completa para assim chegar no entendimento do Todo.

Outra ideia: a aproximação da linguagem de programação para os objetos reais faria com que fosse mais fácil escrever os códigos.

A aproximação da linguagem de programação para os objetos reais tornaria mais precisa a representação do mundo real, tornando mais fácil operar com as características de cada tipo de objeto (classe)

6)Por que que houve a mudança/evolução de linguagem estruturada para linguagem orientada de objeto?

Na linguagem orientada a objetos é mais fácil de reutilizar um código, pois há um relacionamento entre as classes, herança e comportamentos similares de alguns objetos. Isto já não é muito simples de realizar na linguagem estruturada, pois na maioria das vezes para se reutilizar um código teríamos que copiar e colar. Logo, a POO se mostrou necessária tendo em vista que o objetivo final é conseguir reutilizar código para resolver problemas futuros. Fora isso, ele é mais organizado e facilita o controle da privacidade dos dados, devido ao uso dos modificadores de acesso.

GIT GLOSSÁRIO

Clone: Clonar repositório na máquina (ou nuvem).
Status: Comparar com o original e mostrar arquivos separados
Add: Adicionar na branch
Commit: Empacotamento das mudanças e do trabalho desenvolvido. (Pacotes).
Push: Enviar um commit para o repositório local.
Pull: Atualizar o repositório ao invés de clonar.
Merge: Fundir um código ao outro.
Git: é um programa que versiona códigos. (Programa de terminal).

GitHub: é um servidor na nuvem que armazena seus códigos e possui uma interface amigável.

```
$ git branch iss53          = criar uma branch de nome: "iss53".
$ git checkout iss53        = mudar para a branch de nome: "iss53".
$ git merge iss53       = fazer o merge do branch iss53 pro master.
$ git branch -d iss53       = deletar a branch iss53.
```

cardoso: criou o repositorio la no github e pegou a url
abriu a pasta e fez o git bash here
git init
git remote add origin url
git status
foi na pasta e criou o arquivo
git add nome do arquivo
git status
git commit -a -m "mensagem"
git log
git push -u origin master
corri pro abraço

- **Diferença entre ling interpretada e compilada**

Linguagens como C e C++ são *compiladas estaticamente*, e seus códigos fontes são transformados diretamente em *linguagem de máquina*. Enquanto as linguagens mais modernas como Java, C# e Python têm seus códigos fontes transformados em uma *linguagem intermediária* (específica de cada linguagem), que será interpretada pela *máquina virtual da linguagem* quando o programa for executado.

Este processo de interpretação da *linguagem intermediária* durante a execução do programa, consiste na tradução dos comandos da *linguagem intermediária* para *linguagem de máquina*. Sendo assim, em tempo de execução, o código intermediário pode ser encarado como um "código fonte" que será *compilado dinamicamente* pelo *interpretador da linguagem* em código de máquina.