Arquivos fonte utilizados nesta aula:
https://drive.google.com/open?id=1wZKvpF9WLXIqF6XJ_TLfaC4FySRyCghn

Introducao

Nesta aula utilizamos como estudo de caso um programa que contém duas classes muito simples: uma representa Retangulos e outra representa Circulos. Tais classes são derivadas da classe FigBase e possuem um método que, dado um ponto p (em 2D), determina se p está ou não dentro da figura (pontos exatamente na borda da figura não são considerados dentro dela).

O programa testaInclusao.cpp cria 5 figuras geométricas e conta quantas delas contém dois pontos (ponto 0,0 e ponto 15,0). Conforme pode ser observado, ha bugs em algumas das classes e, portanto, os resultados impressos estão errados. Na **Etapa 1** desta aula praticaremos o uso de um depurador (o DDD) para a localização de tais bugs (não tente localizá-los sem o depurador).

Etapa 1

Uma ferramenta que pode ser útil para a localização de bugs são os debuggers ou depuradores.

Nesta aula faremos experimentos com o depurador gdb (alternativamente pode-se utilizar o front-end grafico do GDB, chamado DDD). Informações e instruções sobre o DDD e GDB podem ser obtidas nos seguintes links:

https://www.gnu.org/software/ddd/
https://www.gnu.org/software/gdb/
http://users.ece.utexas.edu/~adnan/gdb-refcard.pdf

Para depurar um programa, e' importante compila-lo utilizando a opção -g do g++ (essa opção armazena junto ao executável do seu programa informações importantes para o depurador, como as linhas de código onde cada comando originalmente estava, nome das variáveis, etc). Após isso, utilize o comando “gdb a.out” (substituindo a.out com o nome do executável do programa).

O GDB (e outros depuradores “step-by-step”) permite que o programador acompanhe a execução do programa passo-a-passo, visualizando os valores das variáveis, o fluxo da execução, etc.

Teste o GDB com o programa fornecido como exemplo para esta aula. O comando “run” executa o programa no depurador.

Não tente modificar o código para depurá-lo (modifique-o apenas para corrigir os bugs). Embora muitas vezes a modificação de software (para, por exemplo, imprimir valores de

variáveis com cout) possa auxiliar o processo de depuração, nesta aula queremos treinar apenas o uso do GDB. Dessa forma, faca toda a depuracao usando o debugger.

Explore os principais comandos do GDB junto com o professor (exemplo: break, run, next,continue, step, display, watch, finish, backtrace, etc). Após ficar familiarizado com esse debugger, use-o para tentar localizar os bugs no código que foi disponibilizado (corrija tais bugs).

Obs: ha vários tipos de debuggers. Por exemplo, um tipo particularmente útil de debugger sao os de memoria (como o Dr. Memory e o Valgrind): tais softwares são capazes de auxiliar o programador a encontrar vazamentos de memória, acesso a posições inválidas de memória (que normalmente causam falhas de segmentação), etc. É recomendado que os alunos pesquisem sobre tais debuggers, pois eles podem ser muito úteis (por exemplo em trabalhos práticos).
Links: http://valgrind.org/docs/manual/quick-start.html , http://drmemory.org/

Etapa 2

Conforme mencionado em sala de aula, o utilitário make pode ser utilizado para facilitar o processo de compilação de código fonte.

Crie um Makefile para gerar o executável do programa testaInclusao.cpp (considere a versao corrigida desse programa). I.e., quando o comando make for utilizado, todas as dependências do programa testaInclusao.cpp devem ser satisfeitas e o executável testaInclusao.out deverá ser gerado. O Makefile deve possuir entradas individuais para cada um dos arquivos .cpp a serem compilados (ou seja, se a modificação de um arquivo deve implicar na recompilação apenas dos arquivos objeto que dependem dele). Além disso, adicione uma entrada “clean” no Makefile (de modo que a chamada ao comando “make clean” delete os arquivos objeto e executáveis gerados pelo make, mantendo apenas o código fonte original do programa e o Makefile).

Submissao da aula pratica:

A solucao deve ser submetida ate as 18 horas da proxima Segunda-Feira utilizando o sistema submitty (submitty.dpi.ufv.br). Atualmente a submissao so pode ser realizada dentro da rede da UFV.

Envie pelo sistema Submitty todos arquivos da Etapa 2. Nesta aula sua solucao não sera avaliada por testes automaticos.