Alexander André de Souza Vieira 13/0039853
Métodos de Programação 02/2018

Trabalho 3

Este trabalho tinha como objetivo contar quantas linhas de código um arquivo em c++ contém. Para isso, optei por abrir o arquivo .c, e através de várias condições(countlines.cpp/countlines.hpp), filtrar as quantidades de linhas em branco, de linhas totais, de comentários no estilo // e no estilo /**/, para no final, obtermos a quantidade efetiva de linhas de código.

Para testar se a contagem realmente estava sendo executada de maneira correta, busquei 4 códigos na internet (Fibonacci.c, rand.c, printasc.c e calculasoma.c), e escrevi 4 testcases diferentes para testar a contagem das linhas em cada ujm dos códigos (testa_countlines.cpp).

Compilar: Navegue até o diretório \source, e execute o comando make.
Executar: Digite o comando ./testa_countlines

Countlines.cpp

Nesse arquivo temos as seguintes funções:

Quant_linhas_efetivas => Recebe como parâmetro o nome do arquivo, abre esse arquivo, e através a chamada das funções conta_linha, conta_linha_em_branco, conta_coment_barra, conta_coment_barra_e_ast, recebe os valores necessários para que seja calculada a quantidade de linhas efetivas de código. Ao final, fecha o arquivo que foi aberto, calcula a quantidade de linhas de código efetivas, e retorna a quantidade de linhas encontradas.

Conta_linha => Recebe como parâmetro o nome do arquivo, abre esse arquivo, e através de um laço de repetição, verifica a quantidade total de linhas presentes nesse arquivo. Ao final, retorna fecha o arquivo que foi aberto, e retorna a quantidade de linhas encontradas.

Conta_linha_em_branco => Recebe como parâmetro o nome do arquivo, abre esse arquivo, e através de laços de repetição e comparativos, verifica a quantidade total de linhas em branco presentes nesse arquivo. Ao final, retorna fecha o arquivo que foi aberto, e retorna a quantidade de linhas encontradas.

Conta_coment_barra => Recebe como parâmetro o nome do arquivo, abre esse arquivo, e através de laços de repetição e comparativos, verifica a quantidade total de linhas de comentários no estilo // estão presentes nesse arquivo. Ao final, retorna fecha o arquivo que foi aberto, e retorna a quantidade de linhas encontradas.

Conta_coment_barra_e_ast => Recebe como parâmetro o nome do arquivo, abre esse arquivo, e através de laços de repetição e comparativos, verifica a quantidade total de linhas de comentários no estilo /**/ estão presentes nesse arquivo. Ao final, retorna fecha o arquivo que foi aberto, e retorna a quantidade de linhas encontradas.

Testa_countlines.cpp

Nesse arquivo temos os seguintes test cases:

```
TEST_CASE("Teste de Contagem de Linhas do Código Fibonacci", "[count_lines_of_code]") {
    REQUIRE(conta_linha(nome_arq) == 38);
    REQUIRE(conta_coment_barra(nome_arq) == 11);
    REQUIRE(conta_linha_em_branco(nome_arq) == 4);
    REQUIRE(conta_coment_barra_e_ast(nome_arq) == 7);
    REQUIRE(quant_linhas_efetivas(nome_arq) == 16);
}

TEST_CASE("Teste de Contagem de Linhas do Código Rand", "[count_lines_of_code]") {
    REQUIRE(conta_linha(nome_arq2) == 42);
    REQUIRE(conta_coment_barra(nome_arq2) == 12);
    REQUIRE(conta_linha_em_branco(nome_arq2) == 16);
    REQUIRE(conta_coment_barra_e_ast(nome_arq2) == 1);
    REQUIRE(quant_linhas_efetivas(nome_arq2) == 13);
}

TEST_CASE("Teste de Contagem de Linhas do Código Printa ASC", "[count_lines_of_code]") {
    REQUIRE(conta_linha(nome_arq3) == 29);
    REQUIRE(conta_coment_barra(nome_arq3) == 14);
    REQUIRE(conta_linha_em_branco(nome_arq3) == 4);
    REQUIRE(conta_coment_barra_e_ast(nome_arq3) == 1);
    REQUIRE(quant_linhas_efetivas(nome_arq3) == 10);
}

TEST_CASE("Teste de Contagem de Linhas do Código Calcula Soma", "[count_lines_of_code]")
{
    REQUIRE(conta_linha(nome_arq4) == 31);
    REQUIRE(conta_coment_barra(nome_arq4) == 12);
    REQUIRE(conta_linha_em_branco(nome_arq4) == 6);
    REQUIRE(conta_coment_barra_e_ast(nome_arq4) == 0);
    REQUIRE(quant_linhas_efetivas(nome_arq4) == 13);
}
```

Onde cada um deles recebe o arquivo a ser aberto correspondente, e faz as assertivas, mostrando que os testes que foram submetidos foram todos aprovados.

Checklist

a) Levantamento de Requisitos

1. Há a necessidade de migração dos dados de sistemas anteriores ?
2. Há a necessidade de compatibilização do novo software com o legado existente?
3. Definição de expectativas de prazos.
4. Definição de custos e disponibilidade de recursos financeiros pelo cliente.
5. Disponibilidade de recursos humanos e tecnológicos.
6. Identificação de riscos técnicos.
7. Há a necessidade de recursos humanos adicionais a serem disponibilizados ?
8. Há a necessidade de aquisição de algum outro software (Banco de Dados, Gerenciador de rede, Linguagem de Programação,..) para que o projeto possa ser desenvolvido e/ou implantado.
9. Necessidade da aquisição de equipamentos para que o projeto seja desenvolvido e/ou implantado.
10. Necessidade de aprovação de lei para que o projeto seja desenvolvido e/ou implantado no cliente.
11. Cada requisito é uma declaração de necessidades, curta e definitiva?
12. Existem condições apropriadas para que o requisito possa existir?

13. Houve a identificação de restrições dos requisitos e do projeto?
14. Os requisitos do produto, restrições do projeto e compromissos assumidos no Projeto Preliminar foram considerados?
15. O requisito é testável?

b) Especificação dos Requisitos

1. Usabilidade – Conjunto de atributos que evidenciam o esforço necessário para se poder utilizar o software, bem como, o julgamento individual desse uso, por um conjunto explícito ou implícito de usuários.
2. Confiabilidade – Conjunto de atributos que evidenciam a capacidade do software de manter seu nível de desempenho sob condições estabelecidas durante um período estabelecido.
3. Eficiência – Conjunto de atributos que evidenciam o relacionamento entre o nível de desempenho do software e a quantidade de recursos usados, sob condições estabelecidas.
4. Portabilidade – Conjunto de atributos que evidenciam a capacidade do software de ser transferido de um ambiente para outro.
5. Manutenibilidade – Conjunto de atributos que evidenciam o esforço necessário para fazer modificações especificadas no software.
6. Todos os requisitos do sistema são realmente requisitos ao invés de soluções de design e implementação?
7. Verificável – É verificável se, e somente se, para cada um dos requisitos contidos no documento, existe um processo finito e economicamente viável através do qual uma pessoa ou máquina possa assegurar que o produto de software atende ao requisito.
8. Modificável – É modificável se, e somente se, modificações possam ser agregadas ao documento de forma fácil, completa e consistente, com relação a estrutura e estilo.
9. Rastreável – É rastreável se, e somente se, a origem de cada um de seus requisitos é clara e a referência a cada um deles é facilitada nos documentos subsequentes do processo ou em uma melhoria da documentação do sistema.
10. Funcionalidade – Conjunto de atributos que evidenciam a existência de um conjunto de funções e suas propriedades especificadas. As funções são as que satisfazem as necessidades explícitas ou implícitas.
11. Mensagem de Erro - Todas as mensagens de erro são únicas e tem significado correto?
12. Correto – É correto se, e somente se, cada requisito expresso for encontrado também no software.
13. Não ambíguo – É não ambíguo se, e somente se, cada requisito declarado seja suscetível a apenas uma interpretação.
14. Completo – É completo se, e somente se, conter toda e apenas a informação necessária para que o software correspondente seja produzido.
15. Consistente – É consistente se, e somente se, nenhum dos requisitos do documento, tomado individualmente, está em conflito com qualquer outro requisito do mesmo documento.

c) Design do Software

1. Todos os padrões de design foram seguidos?
2. O mínimo de dados é passado para cada interface?
3. Um mecanismo de tratamento de erro foi identificado?
4. As estruturas de dados e os nomes dos elementos são facilmente entendidos e seguem uma convenção de nomes?

5. O design de alto nível implementou todos os requisitos?
6. Toda a infraestrutura (backup, recovery, checkpoints) foi tratada?
7. A lógica do programa é correta, completa e clara? Toda a lógica do programa pode ser testada?
8. A chamada de um protocolo segue os padrões do projeto?
9. As especificações externas de cada módulo são completas e testáveis?
10. Todas as funções são claramente especificadas e logicamente independentes?
11. Cada módulo tem baixo acoplamento externo e alta coesão interna?
12. Todos os dados foram definidos e inicializados?
13. O nome dos dados e seus tipos estão em conformidade com o dicionário do projeto?
14. Todos os dados definidos foram usados? Os dados padrões foram usados e estão corretos?
15. As condições de erro foram tratadas de forma não destrutiva? As condições não usuais são tratadas de forma razoável e não destrutiva?

d) Fazer o Código

1. A chamada de um protocolo segue os padrões do projeto?
2. As especificações externas de cada módulo são completas e testáveis?
3. Todas as funções são claramente especificadas e logicamente independentes?
4. Cada módulo tem baixo acoplamento externo e alta coesão interna?
5. As condições de término de loops podem ser realizadas?
6. Todos os dados foram definidos e inicializados?
7. O nome dos dados e seus tipos estão em conformidade com o dicionário do projeto?
8. Todos os dados definidos foram usados? Os dados padrões foram usados e estão corretos?
9. As condições de erro foram tratadas de forma não destrutiva? As condições não usuais são tratadas de forma razoável e não destrutiva?
10. A lógica do programa é correta, completa e clara? Toda a lógica do programa pode ser testada?
11. As estruturas de dados e os nomes dos elementos são facilmente entendidos e seguem uma convenção de nomes?
12. O design de alto nível implementou todos os requisitos?
13. Toda a infraestrutura (backup, recovery, checkpoints) foi tratada?
14. O mínimo de dados é passado para cada interface?
15. Um mecanismo de tratamento de erro foi identificado?

e) Comentar o código e escrever o código com "*design by contract*" com assertivas de entrada, saída, invariantes e como comentários de argumentação do código.

1. Foram definidos os itens, requisitos e funcionalidades que serão testados?
2. Foram definidos o escopo e a abrangência dos testes?
3. A abordagem de teste está clara e atende os requisitos de qualidade?
4. Os testes complementares foram definidos?
5. Os casos especiais são tratados?
6. O ambiente foi especificado de acordo com as necessidades?
7. Precisa ser instalado algum componente?

8. A massa de dados contempla todas as funcionalidades e estruturas internas do módulo?
9. A massa de dados testa valores de fronteira, valores nulos, valores negativos, valores de tipos diferentes e os caminhos independentes do código?
10. Foi testado a consistência dos dados de entrada?
11. Foram testados a necessidade de caixa alta/baixa e a dependência de valores com outros atributos?
12. O código está de acordo com os padrões da empresa?
13. O tratamento de erros e exceções foi incluído no código?
14. Todos os dados foram atualizados corretamente?
15. Foi verificada a solicitação de atualização/exclusão de um dado inexistente?

g) Testar o código

1. O código previne sistematicamente erros de arredondamento?
2. Todos os loops, branches e construções lógicas estão completos, corretos e corretamente aninhados?
3. O código está conforme padrões de codificação pertinentes?
4. O código está bem estruturado, com estilo consistente e consistentemente formatado?
5. Existem procedures que não foram chamadas ou não são necessárias? Existe alguma parte do código que não é executada?
6. Existe alguma parte do código que pode ser alterada por alguma chamada externa de um componente reutilizável ou alguma biblioteca?
7. Todos os case statements tem um default?
8. Existe alguma declaração dentro de um loop que pode ser colocada fora do loop?
9. Todas as variáveis de saída estão atribuídas?
10. Toda a memória alocada é desalocada?
11. Os arquivos são checados se existem antes de se tentar acessá-los?
12. Todas as condições de término dos loops são óbvias e invariavelmente realizáveis?
13. Existe algum bloco de código repetido que pode ser condensado em um procedure?
14. O código é claro e está adequadamente comentado com um estilo de comentário fácil de manter?
15. Todas as variáveis estão definidas de maneira correta e com nomes claros, consistentes e significativas?

h) Depurar o código

1. A massa de dados testa valores de fronteira, valores nulos, valores negativos, valores de tipos diferentes e os caminhos independentes do código?
2. A abordagem de teste está clara e atende os requisitos de qualidade?
3. Foram considerados às restrições ambientais e técnicas do ambiente do sistema?
4. Foram verificadas as restrições de máquina?
5. Os módulos codificam o que está especificado no projeto?
6. As mensagens do programa estão claras?
7. Os requisitos de teste foram determinados conforme a base de prioridades do projeto?
8. Foram definidos o escopo e a abrangência dos testes?
9. Foram testados campos obrigatórios, dígitos verificadores, datas válidas, domínios de tabela, valores de fronteira, valores nulos, valores padrão e de tipos diferentes?

10. Foram feitas simulações para verificar integração com outros sistemas e base de dados?
11. A massa de dados contempla todas as funcionalidades e estruturas internas do módulo?
12. Os testes unitários exercitam todos os cenários possíveis, testando validação de entradas, interações, mensagem de erros, exceções e etc?
13. O teste contém um número limitado de ASSERTS? Um teste unitário bem definido deve conter somente uma declaração Assert.
14. A documentação para que o resultado do teste possa ser verificado foi fornecida?

Foi respeitada a maioria dos itens da checklist, pois a mesma foi seguida desde o início do desenvolvimento do código.