Departamento de Ciência da Computação
Métodos de Programação 1/2016

Projeto de Disciplina

Datas de entrega:

Definição dos grupos: até 25/5/16 23:55

Fase 1 : 14/6/15 23:55

Final : 24/6/16 ate as 23:55

Gerenciador de Tarefas

Descrição :

Faça um programa em C ou C++

O programa deve gerenciar seqüências de tarefas.

O trabalho deve ser feito utilizando a biblioteca de grafos feita no trabalho 2.

O gerenciador é composto de:

a) Leitor e escritor das tarefas em arquivo:

O arquivo deve conter uma tarefa por linha. Os campos são separados por um espaço. Não deve haver espaço entre o último campo e o '\n'.

Os campos são os seguintes:

Id_tarefa nome_tarefa tarefa_executada duração_tarefa inicio_min_tarefa pré_requisitos_tarefa

Id_tarefa: inteiro identificador único da tarefa (cada tarefa pode ter apenas um número e cada número pode ser atribuído apenas a uma tarefa)

nome_tarefa: string tamanho máximo 100 caracteres com o nome da tarefa. A string é tudo entre aspas simples. (as aspas não vão fazer parte da string)

tarefa_executada: inteiro. Se a tarefa já foi executada tem valor 1. Se a tarefa não foi executa tem valor zero. Se a tarefa já foi executada, ela está disponível como pré-requisito mas o tempo dela não deve ser contado para se descobrir o tempo mínimo para completar todas as tarefas.

duração_tarefa: inteiro correspondente a duração da tarefa (em períodos)

inicio_min_tarefa: inteiro que diz o período mínimo em que uma tarefa pode iniciar. Ex: Se este inteiro for 4 a tarefa só pode se iniciar no período 4 em diante. Se for 0 a tarefa pode se iniciar em qualquer período (ou seja, do período 0 em diante).

pré_requisitos_tarefa: o primeiro inteiro diz quantos pré-requisitos existem para a tarefa. A seguir segue um número de inteiros correspondentes as estes pré-requisitos (Id_tarefa)

Ex.

100 ‘primeira tarefa’ 0 3 0 0

101 ‘segunda tarefa’ 0 5 2 0

102 ‘terceira tarefa’ 0 2 3 2 100 101

A linha 1 diz que existe uma tarefa chamada ‘primeira tarefa’ com identificador 100, não foi executada ainda, ela dura 3 períodos, pode começar no período 0 e a sua lista de pré-requisitos é vazia.

A linha 2 diz que existe uma tarefa chama ‘segunda tarefa’ com identificador 101, não foi executada ainda, ela dura 5 períodos e pode começar apenas do período 2 em diante. A sua lista de pré-requisitos é vazia

A linha 3 tem a ‘terceira tarefa’ com identificador 102 e não foi executada ainda. Ela dura 2 períodos e pode começar do período 3 em diante. Ela tem dois pré-requisitos: as tarefas 100 e 101

O tempo é contado em períodos, que correspondem a números inteiros. O período começa do zero.

As tarefas devem começar a executar tão logo seja possível: seus pré-requisitos estejam disponíveis e seu inicio_min_tarefa seja possível.

No exemplo, o tempo mínimo para executar estas tarefas será:

A tarefa 100 começa em 0 e termina em 3.

A tarefa 101 começa em 5 e termina em 7

A tarefa 102 pode começar em 3 mas como os pré-requisitos são a tarefa 100 e 101 ela só pode começar depois do término da tarefa que termina mais tarde que neste caso é a tarefa 101. Ela pode começar então apenas em 7 e termina em 9.

Assim, o tempo mínimo para completar as tarefas e 9.

As tarefas devem poder ser salvas no mesmo formato.

b) Editor de tarefas:

Dever ser possível :

b.1) criar tarefa

b.2) remover tarefa

b.3) editar uma tarefa. Deve se poder editar todos os atributos da tarefa: id_tarefa nome_tarefa tarefa_executada duração_tarefa inicio_min_tarefa pré_requisitos_tarefa

b.4) deve ser verificada a consistência das tarefas. Ex. não pode haver tarefas com id_tarefa repetidos.

c) Visualizador de tarefas:

Deve mostrar visualmente:

c.1) As tarefas

c.2) Os pré-requisitos de cada tarefa

c.3) Os caminhos (seqüências de tarefas) que indicam o menor tempo de execução possível (pode haver mais de um caminho com o mesmo valor).

c.4) Quais tarefas foram completadas ou não com uma cor específica

c.5) Deve ser possível escolher um período e o visualizador mostrar quais são as tarefas completadas até aquele período

c.6) Mostra eventuais inconsistências das tarefas.

d) Avaliador das tarefas:

Esta parte extrai as informações das tarefas que podem ser mostradas no visualizador.

d.1) verifica se as tarefas estão consistentes

d.2) calcula qual é o tempo mínimo necessário para completar todas as tarefas indicando os caminhos para isto.

d.3) diz quais tarefas devem estar completadas em um determinado período

Requisitos:

1) Devem ser aplicados neste trabalho todos os conceitos vistos nos trabalhos anteriores.

a) Modularização (makefile, .h e .c),

b) Testes utilizando o Gtest. Como estes testes mostram que o programa segue a especificação.

c) Assertivas de entrada e de saída. Estas assertivas devem ser colocadas no código através do comando **assert** ou **ifs**. Comentários no código também devem especificar quais são estas assertivas

d) Para cada uma das funções adicionar comentários indicando qual são as interfaces explícita, implícita com a devida descrição, quais são os requisitos e as hipóteses de cada função. Além disto, as funções devem ter finalização adequada liberando memória, fechando arquivos, etc.

e) Faça a modelagem física das estruturas de dados. Use cabeças de estrutura de dados.

f) Assertivas de entrada e de saída como parte da especificação (comentários antes das funções).

g) Assertivas como comentários de argumentação.

h) Assertivas estruturais: elas definem a validade de uma coletânea de dados, ou estruturas de dados, e dos estados associados a estes dados

i) Coloque nos comentários antes das funções quais são as assertivas do **contrato na especificação.** Diga o que deve ser esperado da função cliente em relação à entrada e o que deve ser garantido pela função servidora na saída.

j) Utilize iteradores, programação genérica e ponteiros para função.

2) Instrumente o código usando o gcov. Usando o gcov. (http://gcc.gnu.org/onlinedocs/gcc/Gcov.html). O makefile deve ser modificado de forma incluir as flags -ftest-coverage -fprofile-arcs. Depois de rodar o executável rode gcov nomearquivo e deverá ser gerado um arquivo .gcov com anotação.

O gcov é utilizado para saber qual percentual do código é coberto pelos testes. Neste caso os teste devem cobrir pelo menos 80% do código por módulo.

Faça a análise estática do programa utilizando o *cppcheck*, corrigindo os erros apontados pela ferramenta

Utilize cppcheck --enable=warning
para verificar os avisos

3) Sugestão, utilizar o Valgrind (http://valgrind.org/) . O Valgrind e muito útil para encontrar problemas relacionados a memória. O Valgrind **não** será utilizado na correção dos trabalhos mas pode tornar o desenvolvimento de depuração bem mais rápida.

4) Interface gráfica poderá ser uma destas:

a) **biblioteca ncurses.**
Ela pode ser instalada no terminal shell do Linux usando o comando

> sudo apt-get install libncurses5-dev

Uma vez instalada, a biblioteca ncurses ela pode ser compilada utilizando o GCC e a diretiva –lncurses.
Dependendo da instalação ncurses pode ser incluída por <ncurses/ncurses.h> ou <ncurses.h>.

Existem vários tutoriais disponíveis como :
http://www.tldp.org/HOWTO/NCURSES-Programming-HOWTO/index.html

b) **SDL**

https://www.libsdl.org/

Ela pode ser instalada com apt-get e deve ser devidamente configurada e referenciada no código.

c) **PlayCB**

Ela pode ser instalada a partir de:

http://pt-br.playcb.wikia.com/wiki/Wikia_PlayCB

3) Em um trabalho colaborativo como este, é interessante utilizar o github

https://github.com/

Um tutorial inicial pode ser encontrado em:

https://guides.github.com/activities/hello-world/

4) O tempo em cada tarefa pode ser facilmente contabilizado utilizando aplicativos ou sites como:

https://toggl.com

Módulos

O programa deve ser dividido em módulos. Os módulos implementados nos laboratórios podem ser utilizados aqui.

Observações:

Utilize os princípios de modularidade na criação dos módulos, definidos seus módulos de definição, implementação e organizando suas compilações e ligações (**links**) de forma adequada via um único **makefile**.

Na criação dos módulos, lembre-se de usar as diretivas de controle para evitar inclusões múltiplas e identificando também o módulo servidor.

Controle de qualidade das funcionalidades

Depois de pronto, o aluno deve criar um Makefile. Deve-se criar um *módulo controlador de teste* (disciplinado) usando o Gtest para testar se as principais funcionalidades e restrições dos módulos, atendendo aos critérios da **Interface para linha de Comando**. O teste disciplinado deve seguir os seguintes passos:

1) Antes de testar: produzir um roteiro de teste

    a. definir o contexto (cenário) necessário e selecionar a massa de teste contendo a seqüência de ações e valores de teste com os respectivos resultados esperados e que foi criada segundo um critério de teste.

2) Antes de iniciar o teste: estabelecer o cenário do teste

3) Criar um módulo controlador de teste, usando a ferramenta Gtest para testar as principais funcionalidades de cada módulo.

4) Ao testar: produzir um laudo em que todas as discrepâncias encontradas são registradas. Esse laudo pode ser uma saída da execução do Gtest. Somente termine o teste antes de completar o roteiro, caso observe que não vale mais a pena continuar executando o roteiro, uma vez que o contexto para o resto está danificado.

5) Após a correção: repetir o teste a partir de **2** até o roteiro passar sem encontrar falhas.

Deve ser gerada uma documentação do código usando o programa DoxyGen (http://www.stack.nl/~dimitri/doxygen/): O programa inteiro terá de ser documento usando DoxyGen. Comentários que vão ficar na documentação devem ser do estilo Javadoc. A documentação deverá ser feita conforme visto em aula

Os padrões de codificação e documentação devem estar conforme capítulo 4 e apêndice 4 e 5 do material bibliográfico da disciplina.

Parte 2. Escrita

· Um arquivo **LEIAME.TXT** contendo a explicação de como utilizar o(s) programa(s).
· Tantos arquivos **RELATORIO-nome.TXT** quantos forem os membros do grupo. O tema **nome** deve identificar o membro do grupo ao qual se refere o relatório. Estes arquivos devem conter uma tabela de registro de trabalho organizada como a seguir:
Data | Horas Trabalhadas | Tipo Tarefa | Descrição da Tarefa Realizada

Na descrição da tarefa redija uma explicação breve sobre o que o componente do grupo fez. Esta descrição deve estar de acordo com o Tipo Tarefa. Cada Tipo Tarefa identifica uma natureza de atividade que deverá ser discriminada explicitamente, mesmo que, durante uma mesma sessão de trabalho tenham sido realizadas diversas tarefas. Os tipos de tarefa são:

- estudar aulas e laboratórios relacionados
- especificar os módulos
- especificar as funções
- revisar especificações
- projetar
- revisar projetos
- codificar módulo
- Rodar os CPPCheck e retirar warnings
- revisar código do módulo
- redigir casos de teste
- revisar casos de teste
- realizar os testes
- instrumentar via gcov
- documentar com Doxygen

Observações:

- Dica: Preencha esta tabela de atividades ao longo do processo. NÃO DEIXE PARA ÚLTIMA HORA, POIS VOCÊ NÃO SE LEMBRARÁ DO QUE FEZ TAL DIA, TAL HORA. Com relatórios similares a esse você aprende a planejar o seu trabalho.

O trabalho pode ser feito em grupo de no máximo 3.

O programa deve ser feito para Linux, utilizado o GCC e GDB na mesma versão do Linf

Deve ser enviado um único arquivo compactado (.zip) contendo todos os arquivos necessários (.c .h makefile, estrutura de diretório, informação de como utilizar, etc).

Apenas um integrante do grupo deve enviar o trabalho. O nome do trabalho deve ser algo como:
MP_Jose_12345_Joao_54321_Maria_12345.zip
onde são os primeiros nomes e matriculas dos integrantes do grupo.

Cópias de trabalho terão nota zero.

Excelente trabalho!

Deve ser enviada pelo ead.unb.br até :

Fase 1 : 14/6/16 23:55

Consiste de um programa capaz de ler um arquivo de entrada, mostrar as tarefas e gerar um arquivo de saída

Final: 24/6/16 ate as 23:55

Consiste da Fase 1 e de todo o resto do trabalho