Segunda Prova de Programação I

Data: 12/12/2023
Prof. Flávio Varejão

Todas as soluções das questões desta prova devem ser implementadas em C em um único arquivo chamado p2.c. Coloque seu nome e a identificação da máquina na qual está fazendo a prova como comentários no início deste arquivo.

A Google te contratou para participar do projeto de reengenharia do software Agenda (Calendar) da suite Google. Para tanto, você deve implementar um programa em C para controlar a agenda semanal de eventos dos usuários. Cada evento a ser registrado na agenda contém as seguintes informações: o dia da semana (inteiro no intervalo de 0 a 6, 0 corresponde a domingo, 1 a segunda e assim por diante), a hora de início (inteiro no intervalo de 0 a 23, correspondendo as horas do dia) e a duração do evento (inteiro no intervalo entre 0 e 23). Note que um evento pode durar mais de uma hora, mas tem que ser realizado continuamente. Além disso, nenhum evento pode começar em um dia e terminar no outro (não é necessário verificar essa condição, que sempre será respeitada nos dados de entrada). Você deve:

1. Criar o tipo *tInfoEvento* para armazenar as informações de um evento (0,5 pontos).
2. Criar o tipo *tListaEventos* para armazenar informações sobre os eventos da agenda semanal (0,5 pontos).
3. Fazer uma função chamada *comparaEventos* que receba dois eventos tipo *tInfoEvento* e retorne -1 caso o primeiro evento anteceda o segundo, +1 se o primeiro evento suceder o outro e 0 caso sejam concomitantes. Considere que um evento antecede cronologicamente outro se ele tem seu inicio antes do outro e o sucede caso seu inicio seja depois do outro. Em caso de inícios iguais, os eventos são concomitantes. Considere sempre que o primeiro dia da semana é o domingo (1 ponto).

   int comparaEventos (tInfoEvento e1, tInfoEvento e2)
4. Fazer uma função chamada *incluiEvento* que receba uma lista de eventos ordenada cronologicamente do tipo *tListaEventos* e um evento do tipo *tInfoEvento*. A função deve incluir esse evento na lista mantendo a ordenação cronológica. ATENÇÃO PARA A RESTRIÇÃO: A FUNÇÃO DEVE USAR UM ALGORITMO DE INSERÇÃO ORDENADA. INSERIR O EVENTO NA LISTA EM QUALQUER POSIÇÃO E DEPOIS ORDENAR NÃO É PERMITIDO!!! (1,5 pontos).

   tListaEventos incluiEvento (tListaEventos agenda, tInfoEvento e)
5. Criar o tipo *tMatrizEventos* capaz de armazenar uma matriz com 24 linhas e 7 colunas. Cada coluna representa um dia da semana, começando pelo domingo. Cada linha, representa uma hora do dia, começando as 0 horas e terminando as 23 horas. Cada célula da matriz contém um inteiro. Se a célula da matriz contém o valor 0 (zero) significa que o horário correspondente está vago. Se a célula da matriz contém o valor 1 (um) significa que o horário correspondente está ocupado (0,5 pontos).
6. Fazer uma função chamada *controlaAgenda* que receba uma lista de eventos do tipo *tListaEventos* contendo zero ou mais eventos e retorne uma matriz do tipo *tMatrizEventos* com ocupação registrada de acordo com a lista de eventos recebida pela função (1,5 pontos).

   tMatrizEventos controlaAgenda (tListaEventos agenda)
7. Fazer uma função *leAgenda* para ler a agenda semanal de um usuário de um arquivo texto chamado *"agenda.txt"* e inserir numa lista do tipo *tListaEventos*. Cada linha do arquivo contém informações de um evento na seguinte ordem: dia da semana, hora de início e duração do

evento. A inclusão dos dados na lista de eventos deve ser realizada por ordem cronológica, ou seja, o evento lido deve ser incluído na sequência temporal em que ele ocorre. Para isso, utilize a função *incluiEvento* da questão 4. Não é necessário verificar se um evento conflita com outro. Assuma novamente que os dados de entrada no arquivo *"agenda.txt"* estão todos coerentes. Veja o arquivo "*agenda.txt"* disponibilizado junto a prova como exemplo de formatação (1,5 pontos).

tListaEventos leAgenda (void)

8. Fazer uma função *atualizaAgenda* que receba uma lista de eventos do tipo *tListaEventos* contendo zero ou mais eventos e leia uma série de eventos de um arquivo texto chamado *"atualizacao.txt"*. A função só deve inserir os eventos que não conflitam com os eventos existentes na lista. O arquivo "*atualização.txt"* tem a mesma formatação do arquivo *"agenda.txt"* da questão 7. Note que nessa questão é necessário verificar se o evento a ser inserido não conflita com algum evento já inserido na lista. Para facilitar essa operação, é sugerido o uso da função *controlaAgenda* para verificar se há conflito e da função *incluiEvento* para inserir o evento na agenda. Atente para a necessidade de atualizar a matriz de eventos sempre que um novo evento é incluído na lista (1,5 pontos).

   tListaEventos atualizaAgenda (tListaEventos ag)

9. Fazer uma função *escreveAgenda* que receba uma lista com uma agenda semanal de eventos e gere um arquivo chamado *"atualizada.txt"* contendo os dados da agenda segundo o mesmo formato do arquivo *"agenda.txt"* usado na questão 7 (1 ponto)

   void escreveAgenda (tListaEventos ag)

10. Fazer um programa que leia a agenda semanal do usuário do arquivo *"agenda.txt"*, faça a atualização da agenda conforme dados do arquivo *"atualizacao.txt"* e gere o arquivo *"atualizada.txt"* com os dados da agenda devidamente atualizada (0,5 pontos)

Fazem parte da prova os arquivos anexos *agenda.txt, atualizacao.txt* e *atualizada.txt* com exemplo de formatação para apreciação e teste da sua implementação.

Lembrete - Funções de arquivos:

FILE fopen (char* nomearq, char* modo)*

int fclose (FILE fp)*

*int fscanf (FILE * fp, char* formato, …)*

*int fprintf (FILE * fp, char* formato, …)*

int feof(FILE fp)*

Boa Prova!!!