DCC/ICEx/UFMG — Luigi D. C. Soares
DCC 203 - Programação e Desenv. de Software I — **Segunda Prova - Turma TM3**
2º semestre de 2022 — 08/11/2022

Estudante:	**Matrícula:**

Instruções:

i. Este exame possui 4 página(s). Verifique se sua cópia do exame está completa.

ii. **Escreva seu nome na primeira página das suas respostas** e forneça **todas as suas respostas à mão**.

iii. **Esta prova é sem consulta**: você não tem permissão para consultar o livro texto, suas notas de aula, a Internet, seus colegas ou quaisquer outras pessoas ou fontes para concluir o exame.

iv. Se você acredita que alguma pergunta esteja subespecificada, anote as suposições que você teve que fazer para chegar a sua resposta e justifique-as como parte de sua resposta à pergunta.

v. Códigos devem ser escritos na linguagem C. Caso não se lembre de alguma sintaxe específica, descreva da forma mais clara e sucinta possível o que está querendo fazer.

vi. Lembre-se de indentar o código de maneira apropriada e atente-se a organização, clareza e legibilidade do código!

Este ano tem Copa do Mundo! O país inteiro se prepara para torcer para a equipe canarinho conquistar mais um título, tornando-se hexacampeã. Na Copa do Mundo, depois de uma fase de grupos, dezesseis equipes disputam a Fase final, composta de quinze jogos eliminatórios. A figura abaixo mostra a tabela de jogos da Fase final:

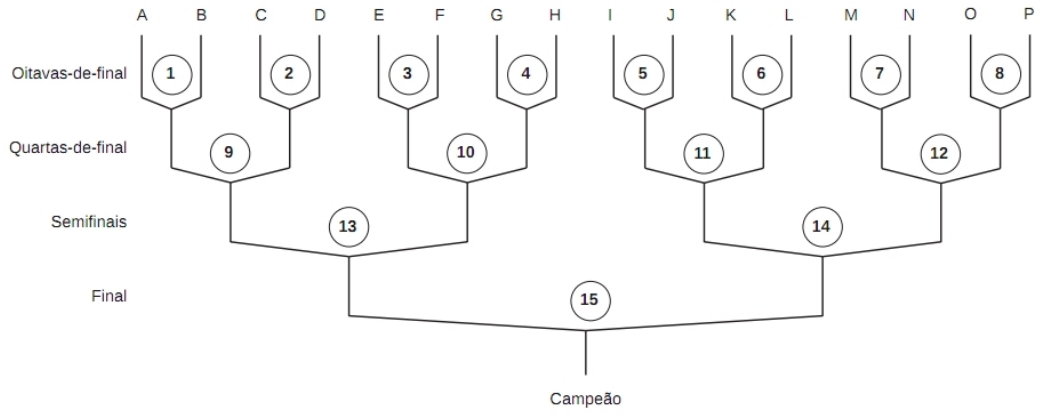

Figura 1: Fonte: `https://br.spoj.com/problems/COPA1/`

Na tabela de jogos acima, as dezesseis equipes finalistas são representadas por letras maiúsculas (de A a P), e os jogos são numerados de 1 a 15. Por exemplo, o jogo 3 é entre as equipes identificadas por E e F; o vencedor desse jogo enfrentará o vencedor do jogo 4, e o perdedor será eliminado. A equipe que vencer os quatro jogos da Fase final será a campeã (por exemplo, para a equipe K ser campeã ela deve vencer os jogos 6, 11, 14 e 15.

Nas questões a seguir, iremos modelar as fases eliminatórias de um campeonato, seguindo um formato similar ao da copa do mundo, representado acima. As seguintes mudanças serão adotadas: (i) a fase inicial terá uma quantidade k de jogos (k múltiplo de 2, $k \leq 64$) e (ii) iremos representar cada time como um inteiro, pois poderemos ter mais de 26 times.

Entrada: A entrada é composta de um inteiro k representando o número de jogos da primeira fase, seguido de $\sum_{i=0}^{\log_2 k} \frac{k}{2^i}$ linhas. Cada linha contém o resultado de um jogo. A primeira linha contém o resultado do jogo de número 1, a segunda linha o resultado do jogo de número 2, e assim por diante. Isto é, as k primeiras linhas contém o resultado dos jogos da primeira fase, as $k/2$ linhas seguintes contém o resultado dos jogos da segunda fase, e assim por diante. A primeira linha contém o resultado do jogo entre os times 0 e 1, a segunda linha contém o resultado do jogo entre os times 2 e 3, e assim

sucessivamente, até acabarem os jogos da primeira fase. A partir da segunda fase, o código dos times que jogam cada fase depende do resultado dos jogos da fase anterior. O resultado de um jogo é representado por dois números inteiros M e N separados por um espaço em branco, indicando respectivamente o número de gols da equipe representada à esquerda e à direita na tabela de jogos ($0 \leq M \leq 20$, $0 \leq N \leq 20$ e $M \neq N$, i.e. vamos desconsiderar empates).

Exemplo:

```
8
4 1
1 0
0 4
3 1
2 3
1 2
2 0
0 2
1 2
4 3
0 1
3 2
3 4
1 4
1 0
```

No primeiro jogo, o time 0 ganhou do time 1. No segundo jogo, o time 2 ganhou do time 3. No terceiro jogo, o time 4 perdeu do time 5. No quarto jogo, o time 6 ganhou do time 7. Consequentemente, o nono jogo foi entre os times 0 e 2 (os vencedores dos dois primeiros jogos) e o décimo jogo foi entre os times 5 e 6. Ao final, o time campeão foi o time 5.

O espaço a seguir é referente ao arquivo de cabeçalho que será utilizado para construir o programa principal na última questão. Utilize este espaço para definir as estruturas descritas nas questões 1 e 2. Você também deve utilizar este espaço para declarar **qualquer** função que venha a implementar, incluindo as funções solicitadas nas questões 3 e 4.

campeonato.h

```
1  #ifndef CAMPEONATO_H_
2  #define CAMPEONATO_H_
3
4  // Questao 01: estrutura "Partida"
5  struct Partida {
6      int timeA;
7      int time B;
8      int golsA;
9      int golsB;
10 }
11
12
13 // Questao 02: estrutura "Fase"
14 struct Fase {
15     struct Partida partidas[64];
16     int num_partidas;
17 }
18
19
20 // Questao 03: declaracao da funcao "vencedor"
21 int vencedor(struct Partida p);
22
23
24 // Questao 04: declaracao da funcao "prox_fase"
25 void prox_fase(struct Fase *atual, struct Fase *prox);
26
27
28 #endif
```

1. (3 pontos) Crie uma estrutura **Partida** para representar uma partida do campeonato. Esta estrutura deve conter:
   - Os campos `timeA` (inteiro) e `timeB` (inteiro), representando os códigos dos times.
   - Os campos `golsA` (inteiro) e `golsB` (inteiro), representando o número de gols feitos por cada time.
2. (3 pontos) Crie uma estrutura **Fase** para representar uma fase do campeonato. Esta estrutura de conter:
   - Um campo `partidas` (arranjo de elementos do tipo `struct Partida`, de tamanho constante 64).
   - Um campo `num_partidas` (inteiro) representando o número efetivo de partidas naquela fase.
3. (5 pontos) Crie uma função `vencedor` que recebe um parâmetro do tipo `struct Partida` e retorna o inteiro que representa o time vencedor da partida. Lembre-se de adicionar a declaração da função no arquivo de cabeçalho.

campeonato.c

```c
#include "campeonato.h"

int vencedor(struct Partida p) {
    if (p.golsA > p.golsB) {
        return p.timeA;
    }

    return p.timeB;
}
```

4. (7 pontos) Escreva uma função `prox_fase` que recebe um primeiro parâmetro `atual` e um segundo parâmetro `prox`. O tipo dos dois parâmetros **deve** ser um **ponteiro** para `struct Fase`. Esta função não tem retorno (i.e. o retorno é `void`). Esta função deve percorrer a fase atual do campeonato, identificando os times vencedores de cada partida, e preencher a próxima fase (i.e. preencher o parâmetro `prox` com as partidas da próxima fase). Lembre-se, também, de atualizar o campo `num_partidas` de `prox` com a quantidade de partidas da fase seguinte. Esta função **não deve** preencher o resultado dos jogos da próxima fase, apenas definir os times de cada partida.

campeonato.c

```c


void prox_fase(struct Fase *atual, struct Fase *prox) {
    prox->num_partidas = atual->num_partidas / 2;
    int i = 0;

    for (int j = 0; j < prox->num_partidas; j++) {
        // atual->partidas eh equivalente a (*atual).partidas (atual eh ponteiro)
        int vencedor1 = vencedor(atual->partidas[i]);
        int vencedor2 = vencedor(atual->partidas[i + 1]);

        // A sintaxe abaixo corresponde a inicializacao de uma
        // estrutura. Os campos sao informados na ordem declarada
        // na definicao da estrutura: timeA, timeB, golsA, golsB.
        // Definimos golsA e golsB com o valor -1 apenas para indicar
        // que a partida ainda nao aconteceu.
        struct Partida p = {vencedor1, vencedor2, -1, -1};
        prox->partidas[j] = p;

        i += 2;
    }
}
```

5. (7 pontos) O programa a seguir irá utilizar as estruturas e funções definidas nas questões anteriores para determinar o vencedor de um campeonato, seguindo a estrutura de entrada de dados definida anteriormente (i.e. cada linha contém o resultado de um jogo). Você pode assumir que as funções implementadas nas questões anteriores estão corretas. O código está parcialmente completo, você deve apenas completar as linhas que estão faltando.

main.c

```
#include <stdio.h>
#include "campeonato.h"

int main(int argc, char *argv[]) {
    int k;
    scanf("%d", &k);

    // Primeiro, vamos construir a fase inicial (k jogos):
    struct Fase inicial;
    inicial.num_partidas = k;

    int timeA = 0, timeB = 1;
    for (int i = 0; i < k; i++) {
        struct Partida p;
        p.timeA = timeA;
        p.timeB = timeB;
        scanf("%d %d", &p.golsA, &p.golsB);

        // Quais sao os identificadores dos dois proximos times?

        timeA += 2_____; timeB += 2_____;

        // Insira a partida na fase inicial:

        inicial.partidas[i] = p______;
    }

    // Em seguida, vamos preencher as fases seguintes:
    // Quantos jogos tem a segunda fase?

    k = k / 2_____________;
    struct Fase atual = inicial;
    while (k > 0) {
        struct Fase prox;
        // Utilize a funcao prox_fase que voce criou para preencher "prox";

        prox_fase(&atual, &prox)_____________________________;

        // Agora vamos ler os resultados dos jogos da proxima fase:
        for (int i = 0; i < prox.num_partidas; i++) {
            scanf("%d %d", &prox.partidas[i].golsA, &prox.partidas[i].golsB);
        }

        // Quantos jogos tem a proxima fase?
        // Dica: o calculo eh o mesmo que para a segunda fase.

        k = k / 2_____________;
        atual = prox;
    }

    //  "atual" sai do loop sendo a ultima fase (final). Imprima o campeao:

    printf("Campeao: %d\n", vencedor(atual.partidas[0]))____________________;
    return 0;
}
```