Na alínea A pretendia-se uma pequena função, para colocar uma peça, para poder ser reutilizada. Quem fez na função main, revela que não tem ainda sentido de reutilização de código, e deve procurar a colocar todo o código operacional fora da função main, devendo na função main serem chamadas as funções e ciclo geral.

```
void MostraPeca(int peca, int posicao)
{
    if (peca > posicao)
        printf("x ");
    else if (peca < -posicao)
        printf("o ");
    else if (posicao == 0)
        printf("_ ");
    else
        printf(". ");
}
```

Em vez de se utilizar as expressões do enunciado, foi optado nesta implementação, colocar-se primeiro as expressões mais simples. Neste caso as últimas expressões são as mais simples, fazendo uso do facto de que a posição nunca é negativa, essas expressões podem simplificar, e as seguintes ficam também mais simples já que são avaliadas apenas após as primeiras. Como resultado tem-se apenas 3 expressões básicas, em vez das 5 expressões lógicas todas com conjunções que está na definição.

Estaria também bem copiar simplesmente as expressões no enunciado, mas houve quem tivesse complicado esta simples função.

Na alínea B era solicitado para imprimir o tabuleiro inteiro. Ao mostrar o tabuleiro, a parte de cima e de baixo originam código parecido mas com o índice invertido. Não foram atribuídos erros de código duplicado parecido nesta situação, dado que é mesmo esta a situação aconselhada, dado que a remoção de todas as duplicações pode originar código demasiado complexo.

```
void MostrarRetiradas(char marca, int retiradas) {
    int i;
    printf("" %c >> "", marca);
    if (retiradas == 0)
        printf(""_"");
    else
        for (i = 0; i < retiradas; i++)
            printf(""%c"", marca);
}

void MostrarTabuleiro(int tabuleiro[24], int barra[2], int retiradas[2])
{
    int i, j;
    // parte de cima
    for (i = 0; i < 5; i++) {
        printf(""\n[ ""); // inicio
        for (j = 12; j < 18; j++)
            MostraPeca(tabuleiro[j], i);
        printf(""] "");
        MostraPeca(barra[0], i);
        printf(""[ "");
        for (j = 18; j < 24; j++)
            MostraPeca(tabuleiro[j], i);
        printf(""]"");
        if (i == 0)
            MostrarRetiradas('o', retiradas[1]);
    }
    printf(""\n"");
```

```
    // parte de baixo
    for (i = 4; i >= 0; i--) {
        printf(""\n[ ""); // inicio
        for (j = 11; j >=6; j--)
            MostraPeca(tabuleiro[j], i);
        printf(""] "");
        MostraPeca(-barra[1], i);
        printf(""[ "");
        for (j = 5; j >= 0; j--)
            MostraPeca(tabuleiro[j], i);
        printf(""]"");
        if (i == 0)
            MostrarRetiradas('x', retiradas[0]);
    }
}
```

Nesta solução separou-se a parte de mostrar as peças retiradas, dado que pode facilmente ser colocado numa função evitando parte da repetição, mas poderia ficar tudo numa só função. Esta é uma alínea B que daria algum trabalho, mas fácil de testar.

Houve quem não tenha identificado a necessidade de ambos os ciclos, e tenha feito esta função por extenso. Se o tabuleiro tivesse uma dimensão variável, ou fosse ainda maior, em vez de 24 casas tivesse 100, ficaria código cada vez maior, e muito dificilmente poderia ser mantido. Quem não utilizou ciclos onde podia, pode procurar a não deixar de fazer um ciclo, nem que sejam três instruções seguidas parecidas, já vale a pena (e mesmo duas).

esta alínea C, naturalmente que não era necessário alterar a estrutura de dados para realizar uma jogada. Isso simplificava a alínea, e era válido, mas apenas faz sentido listar os lances possíveis para o objetivo de os poder executar, e portanto coloco já como versão aconselhada a versão que é útil para a alínea seguinte sem alterações, podendo alterar a estrutura de dados (ou só visualizar as jogadas).

```
void Jogadas(int tabuleiro[24], int barra[2], int retiradas[2],
    int dados[4], char marcas[2], int jogada)
{
    int i, j, k, contador=0;

    if (jogada <= 0) {
        printf(""Dados: "");
        for (j = 0; j < 4 && dados[j]>0; j++)
            printf(""%d "", dados[j]);

        printf(""\nLances: "");
    }

    if (barra[0] > 0) {
        if (JogadaBarra(tabuleiro, barra, dados, jogada, &contador))
            return; // lance efetuado
    }
    else {
        // verificar se todas as peças estão no último quadrante
        for (i = 6, j = 0; i < 24 && j == 0; i++)
            if (tabuleiro[i] > 0)
                j = 1;
        if (j == 0) { // último quadrante, retirar peças
            if (JogadaSair(tabuleiro, barra, retiradas, dados, jogada, &contador))
                return;// lance efetuado
        }
        else { // jogada normal
            if (JogadaNormal(tabuleiro, barra, dados, jogada, &contador))
                return; // lance efetuado
        }
```

```
    }
    if (jogada > 0)
        // jogada inválida, limpar dados
        dados[0] = 0;
}
```

Poderia estar tudo na mesma função, mas atendendo a que há diferentes tipos de jogada conforme o estado do jogo, fica melhor esta divisão, em que há três funções por cada tipo de jogada (ou fase do jogo). Por outro lado, reparem no parâmetro "jogada", que se for 0 pretende-se apenas a listagem das jogadas, se for um número, significa que se pretende efectuar a jogada na estrutura de dados (útil e testado apenas na alínea D).

```
int JogadaNormal(int tabuleiro[24], int barra[2], int dados[4], int jogada,
int *contador)
{
    int i, j;
    for (i = 23; i >= 0; i--)
        if (tabuleiro[i] > 0) { // casa com uma peça, ver se pode jogar
            for (j = 0; j < 4 && dados[j]>0; j++) // analisar todos os dados
                if (i - dados[j] >= 0 && // o dado vai dar a uma casa válida
                    tabuleiro[i - dados[j]] >= -1 && // casa livre, ocupada ou
com no máximo uma peça adversária
                    (j == 0 || dados[j] != dados[j - 1])) // dado distinto do
anterior, para não repetir movimentos
                {
                    if (jogada <= 0) // pretende-se apenas mostrar
                        // lance possível
                        printf("[%d:%d>%d]; ", ++(*contador), i + 1, i -
dados[j] + 1);
                    else if (++(*contador) == jogada) {
                        // efetuar a jogada
                        if (tabuleiro[i - dados[j]] == -1) { // peça
adversária para a barra
                            barra[1]++;
                            tabuleiro[i - dados[j]] = 1;
                        }
                        else
                            tabuleiro[i - dados[j]]++;
                        tabuleiro[i]--;
                        RetirarDado(dados, j);
                        return 1;
                    }
                }
        }
    return 0;
}


int JogadaSair(int tabuleiro[24], int barra[2], int retiradas[2], int
dados[4], int jogada, int *contador)
{
    int i, j, k;
    for (i = 5; i >= 0; i--)
        if (tabuleiro[i] > 0) { // verificar se há uma casa possível
            for (j = 0; j < 4 && dados[j]>0; j++)
                if ((i - dados[j] >= 0 && tabuleiro[i - dados[j]] >= -1 ||
                    i - dados[j] < 0) &&
                    (j == 0 || dados[j] != dados[j - 1]))
```

```
                {
                    // dois dados diferentes podem originar o mesmo movimento
de saída
                    if (i - dados[j] < 0) {
                        for (k = 0; k < j; k++)
                            if (i - dados[k] < 0)
                                break;
                        if (k < j) // movimento já realizado anteriormente
                            continue;
                    }
                    if (jogada <= 0) {
                        // lance possível
                        if (i - dados[j] < 0)
                            printf("[%d:%d>saida]; ", ++(*contador), i + 1);
                        else
                            printf("[%d:%d>%d]; ", ++(*contador), i + 1, i -
dados[j] + 1);
                    }
                    else if (++(*contador) == jogada) {
                        // efetuar a jogada
                        if (i - dados[j] < 0)
                            retiradas[0]++;
                        else {
                            if (tabuleiro[i - dados[j]] == -1) {
                                barra[1]++;
                                tabuleiro[i - dados[j]] = 1;
                            }
                            else
                                tabuleiro[i - dados[j]]++;
                        }
                        tabuleiro[i]--;
                        RetirarDado(dados, j);
                        return 1;
                    }
                }
        }
    return 0;
}



int JogadaBarra(int tabuleiro[24], int barra[2], int dados[4], int jogada, int
*contador)
{
    int j;
    for (j = 0; j < 4 && dados[j]>0; j++)
        if (tabuleiro[24 - dados[j]] >= -1 && // casa destino vazia, positiva
ou apenas com 1 peça adversária
            (j == 0 || dados[j] != dados[j - 1])) // se existirem dados
iguais, não repetir o lance
        {
            if (jogada <= 0) // mostrar lance possível
                printf("[%d:barra>%d]; ", ++(*contador), 24 - dados[j] + 1);
            else if (++(*contador) == jogada) {
                // efetuar a jogada
                if (tabuleiro[24 - dados[j]] == -1) {
```

```
                    barra[1]++;
                    tabuleiro[24 - dados[j]] = 1;
                }
                else
                    tabuleiro[24 - dados[j]] ++;
                barra[0]--;
                RetirarDado(dados, j);
                return 1;
            }
        }
    return 0;
}



void RetirarDado(int dados[4], int i)
{
    while (i < 3 && dados[i]!=0) {
        dados[i] = dados[i + 1];
        i++;
    }
    dados[3] = 0;
}
```

Naturalmente que se removerem o código em que jogada>0, ficaria muito mais simples, mas assim estas funções servem já para a alínea D. Poderiam colocar tudo na mesma função, a mesma ficaria um pouco grande, mas ainda assim dentro do limite.

Nesta alínea até acaba por ser mais fácil que a alínea C. Dois conceitos importantes aqui: lançar os dados, e inverter o tabuleiro. Teria de ter lugar a generalizações da visualização do tabuleiro, e no final verificar vitória, já que os lances já estão feitos. Primeiro, na função main:

```
    while (retiradas[0] < 15 && retiradas[1] < 15) {
        RolarDados(dados);
        while (dados[0] > 0) {
            MostrarTabuleiro(tabuleiro, barra, retiradas, marcas);
            Jogadas(tabuleiro, barra, retiradas, dados, marcas, 0);
            printf("\nJogada %c: ", marcas[0]);
            scanf("%d", &jogada);
            Jogadas(tabuleiro, barra, retiradas, dados, marcas, jogada);
        }
        InverterLados(tabuleiro, barra, retiradas, marcas);
    }
    VerificaVitoria(tabuleiro, barra, retiradas, marcas);
```

A função desenvolvida na alínea C, Jogadas, é chamada duas vezes, com o último argumento 0, para visualizar, e com o número da jogada selecionado, para efetuar o lance.

Acabando de jogar, há que inverter lados, sendo esta uma solução bem mais simples que duplicar código para cada lado a jogar.

A generalização do mostrar peça e tabuleiro é simples, basta adicionar um novo argumento com as marcas a colocar:

```
void MostraPeca(int peca, int posicao, char marcas[2])
{
    if (peca > posicao)
        printf("%c ", marcas[0]);
    else if (peca < -posicao)
```

```
        printf("%c ", marcas[1]);
    else if (posicao == 0)
        printf("_ ");
    else
        printf(". ");
}

void MostrarTabuleiro(int tabuleiro[24], int barra[2], int retiradas[2], char
marcas[2])
{
    int i, j;
    // parte de cima
    for (i = 0; i < 5; i++) {
        printf("\n[ "); // inicio
        for (j = 12; j < 18; j++)
            MostraPeca(tabuleiro[j], i, marcas);
        printf("] ");
        MostraPeca(barra[0], i, marcas);
        printf("[ ");
        for (j = 18; j < 24; j++)
            MostraPeca(tabuleiro[j], i, marcas);
        printf("]");
        if (i == 0) {
            printf(" %c >> ", marcas[1]);
            if (retiradas[1] == 0)
                printf("_");
            else
                for (j = 0; j < retiradas[1]; j++)
                    printf("%c",marcas[1]);
        }
    }
    printf("\n");
    // parte de baixo
    for (i = 4; i >= 0; i--) {
        printf("\n[ "); // inicio
        for (j = 11; j >=6; j--)
            MostraPeca(tabuleiro[j], i, marcas);
        printf("] ");
        MostraPeca(-barra[1], i, marcas);
        printf("[ ");
        for (j = 5; j >= 0; j--)
            MostraPeca(tabuleiro[j], i, marcas);
        printf("]");
        if (i == 0) {
            printf(" %c >> ", marcas[0]);
            if (retiradas[0] == 0)
                printf("_");
            else
                for (j = 0; j < retiradas[0]; j++)
                    printf("%c",marcas[0]);

        }
    }
}
```

A função rolar os dados:

```
void RolarDados(int dados[4])
{
    dados[0] = rand() % 6 + 1;
    dados[1] = rand() % 6 + 1;
    if (dados[0] == dados[1])
        dados[2] = dados[3] = dados[0];
```

```
    else
        dados[2] = dados[3] = 0;
}
```
A inversão é um ponto essencial para simplificar todo o procedimento, sendo possível já que tudo é simétrico, o que ocorre normalmente em jogos entre dois jogadores, de modo a haver equilíbrio:

```
void InverterLados(int tabuleiro[24], int barra[2],
    int retiradas[2], char marcas[2])
{
    int i, j;
    char c;
    for (i = 0; i < 12; i++) {
        j = tabuleiro[i];
        tabuleiro[i] = -tabuleiro[23 - i];
        tabuleiro[23 - i] = -j;
    }
    j = barra[0];
    barra[0] = barra[1];
    barra[1] = j;
    j = retiradas[0];
    retiradas[0] = retiradas[1];
    retiradas[1] = j;
    c = marcas[0];
    marcas[0] = marcas[1];
    marcas[1] = c;
}
```
Finalmente a função final para contabilizar o tipo de vitória:

```
void VerificaVitoria(int tabuleiro[24], int barra[2],
    int retiradas[2], char marcas[2])
{
    static char *descritivo[] = { "vitoria", "grande vitoria", "vitoria enorme" };
    int i, j = 0, pontos = 1;
    if (retiradas[0] == 15) {
        if (retiradas[1] == 0) // grande derrota
            j = 1;
        if (barra[1] > 0) // derrota enorme
            j = 2;
        for (i = 5; i >0 && j==0; i--)
            if (tabuleiro[i] < 0)
                j = 2;
        pontos += j;
        printf("\nGanhou %c com %d pontos (%s)", marcas[0], pontos, descritivo[pontos-
1]);
    }
    else if (retiradas[1] == 15) {
        InverterLados(tabuleiro, barra, retiradas, marcas);
        VerificaVitoria(tabuleiro, barra, retiradas, marcas);
    }
    else
        printf("\nNao ha vitoria de nenhum lado.");
}
```
O código específico da alínea D é até mais simples que o código da alínea C se feito na forma completa.

Ainda uma última proposta de resolução, para o jogador artificial, no caso de haver interessados.

Para fazer o jogador artificial, bastaria definir uma heurística para decidir qual a jogada a fazer, forma de copiar a estrutura de dados para poder testar a heurística nos diversos movimentos, e o jogador artificial o que faz é simplesmente escolher a jogada com a heurística maior.

```
// quantos mais pontos pior para o branco/positivo, é o caminho que este tem de
percorrer
int HeuristicaGamao(int tabuleiro[24], int barra[2],
    int retiradas[2])
{
    int i, j;
    int pontos=0;
    // valorizar peças na posição mais próxima do final
    for (i = 0; i < 24; i++)
        if (tabuleiro[i] > 0)
            pontos += tabuleiro[i] * (i + 1);
        else
            pontos += tabuleiro[i] * (24 - i); // o outro jogador vai para 24
    // as peças que estão retiradas, não penalizam

    // peças que estão fora da última área, contam mais
    for (i = 6; i < 24; i++)
        if (tabuleiro[i] > 0)
            pontos += tabuleiro[i] * (i + 1) * 2;
    for (i = 0; i < 18; i++)
        if (tabuleiro[i] < 0)
            pontos += tabuleiro[i] * (24 - i) * 2;

    // somar um extra pela quantidade de peças não retiradas, ponderado por meio dado
cada
    pontos += (15 - retiradas[0]) * 3 - (15 - retiradas[1]) * 3;

    // somar uma penalização maior, para as peças na barra, que ainda têm de sair,
como custando 2 dados
    pontos += barra[0] * (24 + 12) * 2 - barra[1] * (24 + 12) * 2;

    // peças isoladas contar como risco (24+12)/6=4 pontos a mais
    for (i = 0; i < 24; i++)
        if (tabuleiro[i] == 1)
            pontos += 4;
        else if (tabuleiro[i] == -1)
            pontos -= 4;

    // bonus por casas controladas na sua zona interior
    for (i = 5; i >= 0; i--)
        if (tabuleiro[i] > 1)
            pontos -= 6; // preferivel ter uma casa coberta nesta zona a sair com uma
peça
    for (i = 18; i < 24; i++)
        if (tabuleiro[i] < 1)
            pontos += 6; // preferivel ter uma casa coberta que sair com uma peça

    return pontos; // dividir por 6 para estimar o número de jogadas melhor/pior
}

void Copiar(int tabuleiro[24], int barra[2], int retiradas[2],
    int dados[4], int sentido)
{
    static int tabuleiroB[24], barraB[2], retiradasB[2], dadosB[4];
    int i;
    if (sentido == 0) { // guardar cópia
        for (i = 0; i < 24; i++)
            tabuleiroB[i] = tabuleiro[i];
        for (i = 0; i < 4; i++)
            dadosB[i] = dados[i];
        for (i = 0; i < 2; i++) {
            barraB[i] = barra[i];
```

```
            retiradasB[i] = retiradas[i];
        }
    }
    else { // restaurar cópia
        for (i = 0; i < 24; i++)
            tabuleiro[i] = tabuleiroB[i];
        for (i = 0; i < 4; i++)
            dados[i] = dadosB[i];
        for (i = 0; i < 2; i++) {
            barra[i] = barraB[i];
            retiradas[i] = retiradasB[i];
        }
    }
}

int JogadaArtificial(int tabuleiro[24], int barra[2], int retiradas[2],
    int dados[4], char marcas[2])
{
    int i, totalJogadas, melhorJogada, melhorHeuristica, heuristica;
    // fazer o backup
    Copiar(tabuleiro, barra, retiradas, dados, 0);
    totalJogadas = Jogadas(tabuleiro, barra, retiradas, dados, marcas, 1000);
    Copiar(tabuleiro, barra, retiradas, dados, 1);
    melhorJogada = 1;
    Jogadas(tabuleiro, barra, retiradas, dados, marcas, 1);
    melhorHeuristica = HeuristicaGamao(tabuleiro, barra, retiradas);
    for (i = 2; i <= totalJogadas; i++) {
        Copiar(tabuleiro, barra, retiradas, dados, 1);
        Jogadas(tabuleiro, barra, retiradas, dados, marcas, i);
        heuristica = HeuristicaGamao(tabuleiro, barra, retiradas);
        if (heuristica < melhorHeuristica) {
            melhorHeuristica = heuristica;
            melhorJogada = i;
        }
    }
    Copiar(tabuleiro, barra, retiradas, dados, 1);
    return melhorJogada;
}
```

A função Jogadas teria de ter uma ligeira generalização para poder retornar o número de jogadas válidas, e a função main teria de chamar alternadamente o jogador artificial ou solicitar a jogada ao jogador humano.