Jogo de Cartas War

No jogo de cartas War, um baralho de 52 cartas é distribuído para dois jogadores (1 e 2) de forma que cada jogador fique com 26 cartas. Os jogadores não olham para as suas cartas, mas as mantêm em um monte, viradas com o naipe para baixo. O objetivo do jogo é ganhar todas as cartas do adversário.

Ambos jogam tirando suas cartas de cima do monte e colocando-as em cima da mesa, com sua face virada para cima. Assim, quem vira a carta com mais valor fica com as duas cartas e as adiciona (com a face para baixo) no fundo de seu monte. O valor das cartas, de cima para baixo, são como de costume: A, K, Q, J, T, 9, 8, 7, 6, 5, 4, 3, 2. Nesse jogo, os naipes são ignorados.

Dessa forma, ambos os jogadores repetem esses passos e o jogo continua se repetindo até acabar. O jogo termina quando um jogador ganha todas as cartas.

Quando as cartas viradas para cima são do mesmo valor há uma guerra. Estas cartas permanecem sobre a mesa enquanto os dois jogadores põe sobre a mesa uma carta de seu monte, virada com a face para baixo, e outra virada com a face para cima. Quem tem o maior valor dentre as novas cartas com face para cima ganha a guerra e coloca todos as seis cartas no fundo do seu monte. Se as novas cartas viradas para cima são iguais a guerra continua: cada jogador coloca uma outra carta virada para baixo e uma para cima. A guerra vai continuar enquanto as cartas com face para cima continuarem a ter o mesmo valor. Assim que os valores forem diferentes, o jogador com a carta de mais valor ganha todas as cartas na mesa.

Se alguém ficar sem cartas no meio de uma guerra, o outro jogador já ganha automaticamente. As cartas são adicionados ao monte na ordem exata em que foram jogadas na mesa, especificamente a primeira carta da primeira guerra e a segunda carta da primeira guerra, depois a primeira carta da segunda guerra e a segunda carta da segunda guerra, etc.

Como qualquer pessoa com um sobrinho cinco anos de idade sabe, um jogo de guerra pode levar um longo tempo para terminar. Mas por quanto tempo? O seu trabalho é escrever um programa para simular o jogo e relatar o número de movimentos realizados.

Como podemos ler tal descrição do problema?

Mantenha em mente a ideia de como modelar, escrever, testar e depurar suas soluções:

- Leia cuidadosamente o problema:

  Leia cada linha da demonstração problema com cuidado, e releia-o quando o juiz informar um erro. Leia por alto na primeira vez, pois grande parte da descrição pode ter uma história que não tem impacto sobre a solução. Preste especial atenção às descrições de entrada e saída, e teste a entrada e a saída, mas …
- Sem suposições:

  Ler e entender as especificações é uma parte importante da maratona (e da vida real) de programação. Especificações muitas vezes deixam armadilhas difíceis de encontrar.

  Só porque alguns exemplos apresentam alguma propriedade legal não significa que todos os dados de teste também serão assim. Busque casos não especificados de entrada introduzindo números ilimitados, tamanhos grandes de linhas, números negativos, etc. Qualquer entrada que não é explicitamente proibida deve ser assumida a ser testada!
- Não se preocupe tanto com velocidade de execução:

Eficiência muitas vezes não é uma questão importante, a menos que nós estejamos usando algoritmos exponenciais para problemas onde os algoritmos polinomiais são o suficiente. Não se preocupe muito com a eficiência, a menos que você já tenha visto ou possa prever problemas. Leia a especificação para saber o tamanho máximo de entrada possível e decida se o algoritmo mais simples será suficiente sobre essas especificações.

Apesar do jogo de cartas War parecer interminável quando você está jogando com seu sobrinho (e de fato pode durar para sempre), não vemos nenhuma razão para se preocupar com a eficiência dessa descrição particular problema.

Iniciando a implementação

Qual é a melhor estrutura de dados para representar um baralho de cartas?

A resposta depende do que você vai fazer com eles.

Você vai movê-las? Comparar os seus valores? Procurar padrões do baralho? Suas intenções definem as operações da estrutura de dados.

A ação principal que precisamos aqui é: retirar cartas do topo do monte e colocá-las na parte de baixo dele. Assim, é natural que representar a mão de cada jogador utilizando uma fila FIFO.

Tem um outro problema ainda mais fundamental: como podemos representar uma carta?

Lembre-se que as cartas têm diferentes naipes (ouros, copas, paus e espadas) e valores (as, 2-10, valete, dama, rei).

Temos várias opções, como representar as cartas como um par de caracteres para representar o naipe e o valor. No problema do jogo de cartas War, podemos até nos esquecer dos naipes, mas isso pode dar problemas em outros casos. E se tivéssemos que imprimir o cartão vencedor? Ou se tivéssemos que verificar se a pilha implementada estava funcionando corretamente?

Outra abordagem pode ser representar os cartões com números de 0 à 51 e fazer o mapeamento para a carta quando necessário.

No jogo de cartas War temos que comparar o valor das cartas e isso tornaria a implementação complicada com a representação de caracteres.

Abaixo encontra-se uma implementação útil para classificar as cartas:

```
#define NCARDS 52 /*numero de cartas*/
#define NAIPES 4  /*numero de naipes*/

char valores[] = "23456789TJQKA";
char naipes[]  = "cdhs";

int classificar_carta(char valor, char naipe)
{
    int i,j;

    for ( i = 0; i < (NCARDS/NAIPES); i ++)
        if ( valores[i] == valor )
            for ( j = 0; j < NAIPES; j ++)
                if ( naipes[j] == naipe )
                    return( i*NAIPES + j );

    cout << "Warning: bad input valor="<< valor << ", naipe=" << naipe << endl;
}
```

```
char naipe(int carta)
{
    return( naipes[carta % NAIPES] );
}

char valor(int carta)
{
    return( valores[carta/NAIPES] );
}

int main()
{
    cout << valor(43) << naipe(43) << ' ' << classificar_carta( valor(43),
naipe(43) ) << endl;
}
```

Utilize-o para ler as seguintes entradas (são três rodadas do jogo):

```
4d Ks As 4h Jh 6h Jd Qs Qh 6s 6c 2c Kc 4s Ah 3h Qd 2h 7s 9s 3c 8h Kd 7h Th Td
8d 8c 9c 7c 5d 4c Js Qc 5s Ts Jc Ad 7d Kh Tc 3s 8s 2d 2s 5h 6d Ac 5c 9h 3d 9d
6d 9d 8c 4s Kc 7c 4d Tc Kd 3s 5h 2h Ks 5c 2s Qh 8d 7d 3d Ah Js Jd 4c Jh 6c Qc
9h Qd Qs 9s Ac 8h Td Jc 7s 2d 6s As 4h Ts 6h 2c Kh Th 7h 5s 9c 5d Ad 3h 8s 3c
Ah As 4c 3s 7d Jc 5h 8s Qc Kh Td 3h 5c 9h 8c Qs 3d Ks 4d Kd 6c 6s 7h Qh 3c Jd
2h 8h 7s 2c 5d 7c 2d Tc Jh Ac 9s 9c 5s Qd 4s Js 6d Kc 2s Th 8d 9d 4h Ad 6h Ts
```

```
int main()
{
    char carta[2], c;
    int jogador=0;
    queue<int> montes[2];

    while ( true )
    {
        cin >> noskipws >> carta >> c;
        if (cin.eof()) break;

        cout << carta << '=' << classificar_carta(carta[0], carta[1])
             << " jogador " << jogador << endl;
        montes[jogador].push( classificar_carta(carta[0], carta[1]) );

        if (c == '\n')
        {
            jogador = (jogador+1) % 2;
            cout << "mudou a linha, foi para jogador " << jogador << endl;
            //if (jogador == 0) war( montes );
        }
    }
}
```

Sua tarefa então é implementar o restante desse jogo e realizar os testes.