

# Documentação - Trabalho Prático 2

## Cartas de Baralho

Ryan Junio de Oliveira (5258)

<sup>1</sup>Universidade Federal de Viçosa (UFV)  
Campus Florestal

ryan.oliveira@ufv.br

**Resumo.** O propósito deste trabalho é investigar e reforçar o conteúdo abordado nos módulos dois e três da disciplina, que abordam tópicos sobre Pilhas/Filas, Algoritmos de Ordenação e Análise de Complexidade, através de um jogo de cartas. Em que foi explorado a lógica para construção do jogo com qualquer tipo de entrada.

### 1. Introdução

O programa é um jogo de baralho que permite a manipulação e jogabilidade de cartas. Ele oferece funcionalidades de ordenação das cartas além de possibilitar jogos de cartas entre os jogadores. O sistema também inclui recursos de embaralhamento proporcionando uma experiência completa de baralho.

No jogo, o objetivo é atingir uma sequência de cartas de mesmo valor e, o jogador que conseguir tal feito, ganha. Dentre suas maiores dificuldades está na verificação da mão do jogador, geração de cartas e a dinâmica de implementação. Na ordenação, a maior dificuldade está em comparar naipe e valor, visto que os naipes tem uma ordem de prioridade e a quantidade grande de cartas.

O funcionamento do código está no tipo abstrato de dados, em que é dividido pelo arquivo tad\_baralho.h, tad\_baralho.c e main.c. Isso é, definição de struct e funções, desenvolvimento das funções e suas lógicas e o desenvolvimento do jogo, respectivamente.

### 2. Modelagem e Funcionamento

O jogo baseia-se em poder comprar uma carta do baralho ou capturar uma carta do topo, que havia sido descartada. Para isso, há uma variável chamada "primeiraJogada" que carrega o valor True, após a tentativa do jogador um, essa variável recebe False.

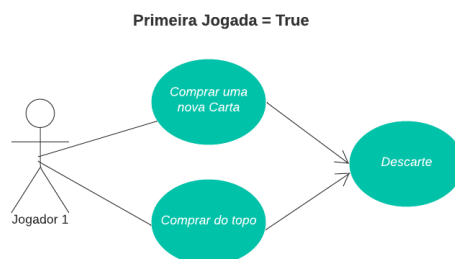
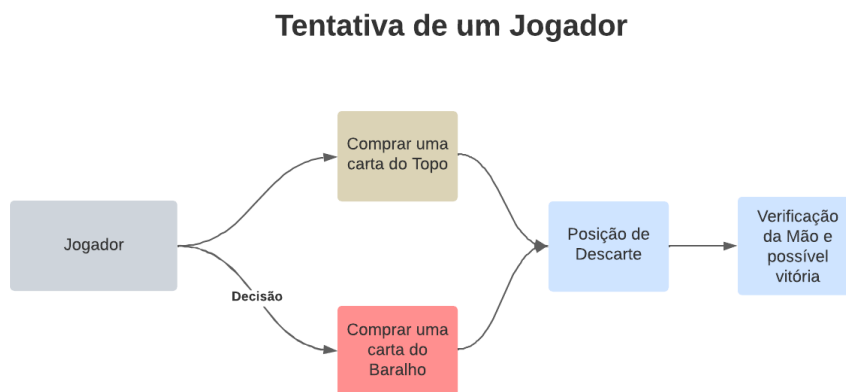


Figura 1. Diagrama da Primeira Jogada

Essa estratégia é usada para que haja um determinado começo, tendo em vista que no começo do jogo não há nenhuma carta no descarte, dessa forma, o primeiro jogador tem apenas a opção de comprar a carta e descartar para o segundo jogador. Após a primeira jogada, acontece um ciclo contínuo até que uma mão vença, sendo esse o critério para encerrar o looping.



**Figura 2. Mapa Conceitual de uma tentativa**

### 3. Implementação

#### 3.1. Leitura do Arquivo

De início temos a abertura do arquivo de forma tradicional, declarando o FILE e abrindo com a condição de que encerrar o procedimento se o arquivo não for aberto corretamente.

```

15 //ARQUIVO DE ENTRADA
16 FILE *arquivo_entrada = fopen("entrada_buraco.txt", "r");
17 if (arquivo_entrada == NULL) {
18     printf("Não foi possível abrir o arquivo.\n");
19     return 1;
20 }
21
22 //ARQUIVO DE SAIDA
23 FILE* arquivo_saida;
24 sprintf(nome_arquivo, "ordenacao_entrada_%s.txt", metodo);
25
26 arquivo_saida = fopen(nome_arquivo, "w");
27 if(arquivo_saida == NULL){
28     printf("Erro ao criar o arquivo.\n");
29     return 1;
30 }
  
```

**Figura 3. Declaração e Leitura do arquivo**

Após isso, começamos a ler cada linha da entrada.txt e fazendo suas determinadas operações.

```

33 // LEITURA DO ARQUIVO
34 char linha[100];
35
36 //Numero de Baralhos
37 fgets(linha, sizeof(linha), arquivo_entrada);
38 sscanf(linha, "%d", &nBaralho);
39 nCartas = quantidadeCartas(nBaralho);
40
  
```

**Figura 4. Número de Baralhos**

Para receber os valores excluídos utilizei da estratégia do strtok, que separa os caracteres com base na referência que colocamos. Neste caso, foi utilizado a vírgula. É importante ressaltar que, através dessa lógica de armazenar no vetor, podemos receber qualquer quantidade de cartas excluídas.

```
41 //Cartas Excluídas
42 fgets(linha, sizeof(linha), arquivo_entrada);
43 vExcluidos nExcluidos;
44 nExcluidos.qValores = 0;
45 char *token = strtok(linha, ",");
46 while(token != NULL){
47     nExcluidos.valores[nExcluidos.qValores] = atoi(token);
48     nExcluidos.qValores++;
49     token = strtok(NULL, ",");
50 }
```

**Figura 5. Cartas Excluídas**

A carta coringa é colocado em um array, em que seus caracteres são acessados de forma individual. Em que, há uma comparação pra ser as posições 0 e 1 são diferentes de 0, dessa forma, há a comprovação de que existe ou não uma carta coringa. Se caso existe, a variável "existeCoringa" recebe True. Ademais, existe algumas comparações para modificar os valores para os inteiros, facilitando a manipulação da carta coringa no código.

```
53 fgets(linha, sizeof(linha), arquivo_entrada);
54 sscanf(linha, "%s", &cartaCoringaLinha);
55 Carta cartaCoringa;
56 int existeCoringa = False;
57 if(cartacoringalinha[0] != 0 && cartacoringalinha[1] != 0){
58     if(cartacoringalinha[0] != 'A' && cartacoringalinha[0] != 'J' &&
59        cartacoringalinha[0] != 'Q' && cartacoringalinha[0] != 'K'){
60         int aux = atoi(&cartacoringalinha[0]);
61         cartaCoringa.valor = aux;
62     }
63     else{
64         if(cartacoringalinha[0] == 'A'){
65             cartaCoringa.valor = 1;
66         }
67         else if(cartacoringalinha[0] == 'J'){
68             cartaCoringa.valor = 11;
69         }
70         else if(cartacoringalinha[0] == 'Q'){
71             cartaCoringa.valor = 12;
72         }
73         else if(cartacoringalinha[0] == 'K'){
74             cartaCoringa.valor = 13;
75         }
76     }
77     cartaCoringa.naipes = cartacoringalinha[1];
78     existeCoringa = True;
79 }
```

**Figura 6. Carta Coringa**

A quantidade de cartas na mão é bem simples, só recebemos ela com a função fgets e colocamos dentro da variável.

```
80 //Quantidade de Cartas na Mão
81 fgets(linha, sizeof(linha), arquivo_entrada);
82 sscanf(linha, "%d", &mao);
```

**Figura 7. Mão**

### 3.2. Struct

Temos de início temos dois structs que definem o nosso baralho. São muito usados dentre todo o código, sendo de extrema importância.

```

13  ▾ typedef struct{
14      |   char naipe;
15      |   int valor;
16      | }Carta;
17
18  typedef int Apontador;
19
20  ▾ typedef struct{
21      |   Carta carta[MAX_ITEM_VETOR];
22      |   Apontador Topo;
23      |   int quantidade_cartas;
24      | }Baralho;
25

```

**Figura 8. Struct Pilha do Baralho**

Logo após, identifiquei a necessidade de implementar um struct para as cartas excluídas, caso haja. Isso facilitaria a manipulação delas dentro do código.

```

26  ▾ typedef struct{
27      |   int valores[MAX_ITEM];
28      |   int qValores;
29      | }vExcluidos;

```

**Figura 9. Struct Valores Excluídos**

## 4. Funções

Há 12 funções para o funcionamento do jogo, essa quantidade exemplifica a complexidade do código e uma maneira de otimizar seu desempenho. A construção do jogo é baseada em pilhas, o que significa que muitas dessas funções estão relacionadas à criação e manipulação das pilhas.

### 4.1. Função: criarBaralho

A função recebe como parâmetros um ponteiro para o baralho a ser criado (pBaralho), o número de baralhos a serem criados (nBaralho), uma estrutura que contém informações sobre as cartas a serem excluídas (vExcluidos), e um vetor de cartas (vBaralho) onde as cartas serão armazenadas.

```

46  void criarBaralho(Baralho* pBaralho, int nBaralho, vExcluidos nExcluidos, Carta vBaralho[MAX_ITEM]){
47      srand(time(NULL));
48      Baralho* pBaralhoAux = malloc(sizeof(Baralho));
49      Carta* cartaAux = malloc(sizeof(Carta));
50      int vetorValor[13] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
51      char vetorNaipes[4] = {'C', 'O', 'P', 'E'};
52      int qCartas = quantidadeCartas(nBaralho);

```

**Figura 10. Parâmetros e Definição de Variáveis**

A função utiliza a função "srand" para inicializar a semente do gerador de números aleatórios com base no tempo atual. Em seguida, são alocados dois ponteiros: pBaralhoAux, que será usado para manipular temporariamente o baralho durante a criação, e

cartaAux, que será usado para manipular temporariamente cada carta durante o processo. É importante ressaltar que, o baralho é criado em uma pilha e em vetor, com intuito de facilitar a ordenação.

Em seguida, são definidos dois vetores: vetorValor, contendo os valores das cartas (1 a 13), e vetorNaipes, contendo os naipes das cartas (C, O, P, E).

Um loop é executado para criar cada carta do baralho. Dentro do loop, as variáveis iValor e iNaipes são calculadas com base no índice atual, usando o operador de módulo para garantir que elas estejam dentro do intervalo correto. A carta é alocada dinamicamente e preenchida com o naipe e valor correspondentes. Em seguida, a carta é adicionada ao baralho auxiliar (pBaralhoAux) usando a função "Empilha".

```
54  for(int i=0; i<qCartas; i++){
55      int iValor = i % 13;
56      int iNaipes = i % 4;
57      Carta* carta = malloc(sizeof(Carta));
58      carta->naipes = vetorNaipes[iNaipes];
59      carta->valor = vetorValor[iValor];
60      vBaralho[i].naipes = vetorNaipes[iNaipes];
61      vBaralho[i].valor = vetorValor[iValor];
62      Empilha(pBaralhoAux, carta, nBaralho);
63  }
```

**Figura 11. Cartas Aleatórias**

Após a criação de todas as cartas, outro loop é executado para verificar se alguma das cartas criadas deve ser excluída, com base nas informações contidas na estrutura nExcluidos. O loop percorre cada carta do baralho auxiliar (pBaralhoAux) e verifica se o valor da carta corresponde a algum valor excluído na estrutura nExcluidos. Se a carta não for excluída, ela é copiada para a variável cartaAux e adicionada ao baralho final (pBaralho) usando a função "Empilha".

```
66  int cont=0;
67  for(int i=0; i<qCartas; i++){
68      cont = 0;
69      for(int j=0; j<nExcluidos.qValores; j++){
70          if(pBaralhoAux->carta[i].valor == nExcluidos.valores[j]){
71              cont++;
72          }
73      }
74      if(cont == 0){
75          cartaAux->naipes = pBaralhoAux->carta[i].naipes;
76          cartaAux->valor = pBaralhoAux->carta[i].valor;
77          Empilha(pBaralho, cartaAux, nBaralho);
78      }
79  }
```

**Figura 12. Cartas Excluídas**

O algoritmo de embaralhamento percorre o baralho de cartas e troca aleatoriamente as posições das cartas. Isso é feito através de um loop que percorre o baralho

do fim ao início. Para cada posição, é gerado um número aleatório  $j$  dentro do intervalo válido, e as cartas nas posições  $i$  e  $j$  são trocadas de lugar tanto no baralho principal (`pBaralho->carta`) quanto no vetor de cartas auxiliar (`vBaralho`). Isso garante que as cartas fiquem embaralhadas de forma aleatória

```
81 //Algoritmo de Embaralhamento
82 for(int i=(qCartas-1)-(nExcluidos.qValores*4); i>0; i--){
83     int j = rand() % (i + 1);
84     Carta temp = pBaralho->carta[i];
85     pBaralho->carta[i] = pBaralho->carta[j];
86     pBaralho->carta[j] = temp;
87
88     Carta temp2 = vBaralho[i];
89     vBaralho[i] = vBaralho[j];
90     vBaralho[j] = temp2;
91 }
92
93 free(pBaralhoAux);
94 free(cartasAux);
95
96 }
```

Figura 13. Embaralhamento

#### 4.2. Função: descarteCarta

No funcionamento do jogo, é necessário que o jogador descarte uma carta. A função utiliza uma lógica simples, ela percorre as cartas da mão uma por uma, utilizando um loop, e verifica se cada carta é a carta que deve ser descartada. Se for a carta correta, ela é adicionada ao baralho de descarte. Caso contrário, a carta é temporariamente armazenada em um baralho auxiliar. Após percorrer todas as cartas, as cartas restantes do baralho auxiliar são transferidas de volta para a mão do jogador. Essa lógica garante que apenas a carta desejada seja descartada, mantendo as outras cartas na mão do jogador.

```
154 void descarteCarta(Baralho* pJogador, int mao, int iDescarte, Baralho* pDescarte, int nBaralho){
155     Baralho* pAux = malloc(sizeof(Baralho));
156     criarPilha(pAux);
157     Carta* cartasAux = malloc(sizeof(Carta));
158     for(int i=0; i<mao+1; i++){
159         Desempilha(pJogador, cartasAux);
160         if(iDescarte == i){
161             Empilha(pDescarte, cartasAux, nBaralho);
162         }
163         else{
164             Empilha(pAux, cartasAux, nBaralho);
165         }
166     }
167     for(int i=0; i<mao; i++){
168         Desempilha(pAux, cartasAux);
169         Empilha(pJogador, cartasAux, nBaralho);
170     }
171 }
```

Figura 14. Função descarteCarta

#### 4.3. Função: tentativaJogo

Essa função tem como objetivo otimizar as tentativas dos jogadores, visto que são etapas que acontecem de forma repetitiva.

A função começa imprimindo a mão do jogador usando a função `imprimeCartas`. Em seguida, exibe a carta do topo do baralho de descarte e solicita ao jogador que faça uma decisão: escolher a carta do topo do descarte (0) ou uma carta do baralho principal (1).

```
173 void tentativaJogo(Balaho* pBaralho, Balaho* pJogador, Balaho* pDescarte, int mao, int nBaralho){
174     //IMPRESSÃO DA MÃO DO JOGADOR
175     printf("SUA MAO: ");
176     imprimeCartas(pJogador, mao-1);
177
178     //IMPRIMIR CARTA DO TOPO E DECISAO DO JOGADOR
179     int iDecisao=0;
180     int iDescarte;
181     printf("CARTA TOPO: ");
182     imprimeCarta(pDescarte->carta[pDescarte->Topo-1]);
183     printf("CARTA TOPO (0) OU CARTA BARALHO (1): ");
184     scanf("%d",&iDecisao);
```

**Figura 15. Decisão do Jogador**

Se o jogador escolher a carta do topo do descarte (decisão 0), a função desempilha essa carta do baralho de descarte, armazena em uma variável auxiliar `cartaAux` e empilha essa carta no baralho do jogador. Em seguida, a função imprime a mão atualizada do jogador, solicita a posição da carta a ser descartada e chama a função `descarteCarta` para descartar a carta escolhida no baralho de descarte. Por fim, imprime a mão do jogador atualizada.

```
186     if(iDecisao == 0){
187         Carta* cartaAux = malloc(sizeof(Carta));
188         Desempilha(pDescarte, cartaAux);
189         Empilha(pJogador, cartaAux, nBaralho);
190         printf("SUA MAO: ");
191         imprimeCartas(pJogador, mao);
192         printf("POSICAO DE DESCARTE: ");
193         scanf("%d",&iDescarte);
194         descarteCarta(pJogador, mao, iDescarte, pDescarte, nBaralho);
195         printf("MAO ATUALIZADA: ");
196         imprimeCartas(pJogador, mao-1);
197     }
```

**Figura 16. Decisão 0**

Se o jogador escolher comprar uma carta do baralho principal (decisão 1), a função desempilha uma carta do baralho principal, armazena em `cartaDesempilha` e empilha essa carta no baralho do jogador. Em seguida, imprime a mão atualizada do jogador, solicita a posição da carta a ser descartada e chama a função `descarteCarta` para descartar a carta escolhida no baralho de descarte. Por fim, imprime a mão do jogador atualizada.

```

198  else if(iDecisao == 1){
199      Carta* cartaDesempilha = malloc(sizeof(Carta));
200      Desempilha(pBaralho, cartaDesempilha);
201      Empilha(pJogador, cartaDesempilha, nBaralho);
202      imprimeCartas(pJogador, mao);
203      printf("POSICAO DE DESCARTE: ");
204      scanf("%d",&iDescarte);
205      descarteCarta(pJogador, mao, iDescarte, pDescarte, nBaralho);
206      printf("MAO ATUALIZADA: ");
207      imprimeCartas(pJogador, mao-1);
208  }

```

Figura 17. Decisão 1

#### 4.4. Função: verificaGanhador

A função verificaGanhador é responsável por verificar se todas as cartas na mão do jogador têm o mesmo valor. A função começa declarando duas variáveis, igual e maoJogador, ambas inicializadas com zero. Em seguida, ela percorre a mão do jogador e armazena os valores das cartas em um array chamado maoJogador.

Após armazenar os valores, a função realiza um segundo loop para comparar o primeiro valor com todos os outros valores no array maoJogador. Cada vez que encontra um valor igual ao primeiro, incrementa a variável igual.

Por fim, a função verifica se o número de cartas na mão (mao) é igual ao valor de igual. Se forem iguais, isso significa que todas as cartas têm o mesmo valor. Nesse caso, a função retorna o valor 1, indicando que o jogador ganhou.

```

98  int verificaGanhador(Baralho* pJogador, int mao, Carta cartaCoringa){
99      int igual = 0;
100     int maoJogador[3];
101     for(int i=0; i<mao; i++){
102         maoJogador[i] = pJogador->carta[i].valor;
103     }
104     for(int i=0; i<mao; i++){
105         if(maoJogador[0] == maoJogador[i]){
106             igual++;
107         }
108     }
109     if(mao == igual){
110         int i=1;
111         return i;
112     }
113 }

```

Figura 18. Decisão 1

#### 4.5. Funções Auxiliares

Dentre todas as funções, é necessário redigir sobre as auxiliares que são implementadas. Elas tem como objetivo otimizar o código. Para o uso das cartas, foi utilizado funções como: "quantidadeCartas", "imprimeCarta" e "imprimeCartas".

Se tratando de pilhas, foi necessário funções que contribuíam para a criação e manipulação das mesmas. Entretanto, não fugiu do que trabalhamos em sala, dessa forma,



foi alterado dentro do nosso contexto, ou seja, adaptando para nosso struct. São elas: "Empilha", "Desempilha", "criarPilha", "verificaTopoPilha" e "verificaPilhaVazia"

## **5. Estudo de Complexidade**

Para o Estudo de Complexidade, irei abordar as funções mais importantes do código, em que é necessário um foco maior.

### **5.1. Ordem de Complexidade: criarBaralho**

Preencher vBaralho e empilhar cartas em pBaralhoAux: O loop for executa qCartas vezes, onde qCartas é igual a quantidadeCartas(nBaralho). Portanto, a complexidade é  $O(qCartas)$ .

Verificar se cada carta em pBaralhoAux está na lista de exclusões: O loop for executa qCartas vezes. No loop interno, são feitas no máximo nExcluidos.qValores comparações. Portanto, a complexidade é  $O(qCartas * nExcluidos.qValores)$ .

Empilhar cartas em pBaralho: O loop for executa qCartas vezes. Portanto, a complexidade é  $O(qCartas)$ .

Algoritmo de Embaralhamento: O loop for executa  $(qCartas - 1) - (nExcluidos.qValores * 4)$  vezes. Sendo assim, complexidade é  $O(qCartas)$ .

### **5.2. Ordem de Complexidade: verificaGanhador**

Criar um array maoJogador e atribuir os valores das cartas da mão do jogador: O loop for executa mao vezes. Portanto, a complexidade é  $O(mao)$ .

Verificar se todos os valores em maoJogador são iguais: O loop for executa mao vezes. Dessa forma,  $O(mao)$ .

### **5.3. Ordem de Complexidade: descarteCarta**

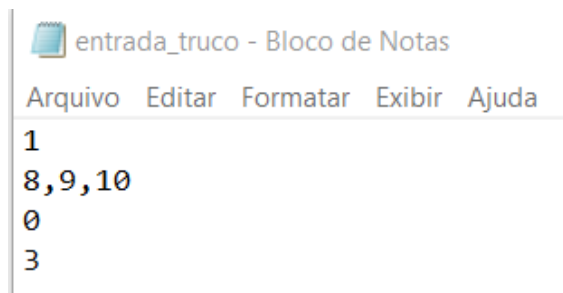
Desempilhar cartas do jogador e decidir se empilhar no descarte ou em pAux: O loop for executa mao+1 vezes. No pior caso, todas as cartas serão desempilhadas e uma decisão será tomada para cada uma delas. Portanto, a complexidade é  $O(mao)$ .

Desempilhar cartas de pAux e empilhá-las de volta no jogador: O loop for executa mao vezes, onde mao é o tamanho da mão do jogador. Portanto, a complexidade é  $O(mao)$ .

## **6. Testes e Resultados**

### **6.1. Jogo**

Como descrito acima, o jogo é definido em etapas, para o teste, estarei utilizando do arquivo de entrada "Truco". Em que há um baralho, cartas excluídas e a mão de três cartas.



**Figura 19. Entrada Truco**

```
PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL

--- INICIO DO JOGO ---
Cada jogador devera fazer uma sequencia de 3 cartas.

--- JOGADOR UM ---
SUA MAO: [4 O] [4 C] [K C] [6 O]
POSICAO DE DESCARTE: 2
MAO ATUALIZADA: [4 O] [4 C] [6 O]
```

**Figura 20. Primeira Jogada - Jogador 1**

```
--- JOGADOR DOIS ---
SUA MAO: [3 P] [K P] [4 E]
CARTA TOPO: [K C]
CARTA TOPO (0) OU CARTA BARALHO (1): 0
SUA MAO: [K C] [3 P] [K P] [4 E]
POSICAO DE DESCARTE: 3
MAO ATUALIZADA: [K C] [3 P] [K P]
```

**Figura 21. Tentativa Jogador 2**

```
--- JOGADOR UM ---
SUA MAO: [4 O] [4 C] [6 O]
CARTA TOPO: [4 E]
CARTA TOPO (0) OU CARTA BARALHO (1): 0
SUA MAO: [4 E] [4 O] [4 C] [6 O]
POSICAO DE DESCARTE: 3
MAO ATUALIZADA: [4 E] [4 O] [4 C]

JOGADOR UM GANHOU!
MAO VENCEDORA: [4 E] [4 O] [4 C]
```

**Figura 22. Tentativa Jogador 1 (VENCEDOR)**

```
--- INICIO DO JOGO ---
Cada jogador devera fazer uma sequencia de 6 cartas.

CARTA CORINGA: [A E]
--- JOGADOR UM ---
SUA MAO: [4 E] [A E] [J E] [6 E] [Q O] [9 E] [9 O]
POSICAO DE DESCARTE: 0
```

**Figura 23. Entrada Genérica (Proposta no Enunciado)**

## 7. Algoritmos de Ordenação

A ordenação proposta no enunciado, exigia o Quicksort e mais outros dois algoritmos de ordenação, em que escolhi o Selectionsort e Bubblesort. Em todos os códigos foram utilizados uma função chamada comparaCartas.

A função ComparaCartas recebe duas cartas e compara seus valores e naipes para determinar a ordem de classificação entre elas. Ela utiliza um mapeamento de naipes para números, em que a ordem de complexidade deste código é constante, ou seja,  $O(1)$ . Segue a seguinte lógica:

```
267 int ComparaCartas(Carta carta1, Carta carta2) {
268     int mapping[256];
269     mapping['C'] = 1;
270     mapping['O'] = 2;
271     mapping['P'] = 3;
272     mapping['E'] = 4;
273 }
```

Figura 24. Mapping

1. Compara os valores das cartas. Se o valor da primeira carta for menor, retorna -1. Se for maior, retorna 1.

```
274 // Compara os naipes utilizando o mapeamento
275 if(carta1.valor < carta2.valor){
276     return -1;
277 }else if(carta1.valor > carta2.valor){
278     return 1;
279 }
280 }
```

Figura 25. Primeira Comparação

2. Se os valores forem iguais, compara os naipes usando o mapeamento. Se o naipe da primeira carta for menor, retorna -1. Se for maior, retorna 1.

```
281 // Se os valores forem iguais, compara os naipes
282 if(mapping[carta1.naipe] < mapping[carta2.naipe]){
283     return -1;
284 }else if(mapping[carta1.naipe] > mapping[carta2.naipe]){
285     return 1;
286 }
287
288 // Se os naipes também forem iguais, retorna 0
289 return 0;
290 }
```

Figura 26. Segunda e Terceira comparação

3. Se os naipes também forem iguais, retorna 0 para indicar que as cartas são consideradas iguais.

### 7.1. QuickSort

No quicksort, trabalhamos com duas funções além propriamente do Quicksort, são elas: Particao e Ordena. A função QuickSort é a função que chama a função Ordena com os parâmetros adequados para ordenar o vetor de cartas A. O parâmetro n indica o número de elementos no vetor.

```

235 void QuickSort(Carta A[MAX_ITEM_VETOR], int n){
236     Ordena(0, n-1, A);
237 }

```

**Figura 27. Função: Quicksort**

A complexidade do Quicksort no pior caso é  $O(n^2)$ , mas no caso médio é  $O(n \log n)$ , onde  $n$  é o tamanho do vetor.

### 7.1.1. Função: Particao

A função Particao é responsável por realizar a partição do vetor. Ela recebe como parâmetros o índice da posição inicial (Esq) e final (Dir) do subvetor a ser particionado, os ponteiros  $i$  e  $j$ , que serão atualizados para indicar as posições de divisão do vetor, e o array  $A$  contendo as cartas.

```

212 void Particao(int Esq, int Dir, int *i, int *j, Carta A[MAX_ITEM_VETOR]){
213     Carta pivo, aux;
214     *i = Esq; *j = Dir;
215     pivo = A[( *i + *j)/2]; /* obtem o pivo x */
216     do{
217         while (ComparaCartas(pivo, A[*i]) > 0) (*i)++;
218         while (ComparaCartas(pivo, A[*j]) < 0) (*j)--;
219         if (*i <= *j)
220         {
221             aux = A[*i]; A[*i] = A[*j]; A[*j] = aux;
222             (*i)++; (*j)--;
223         }
224     } while (*i <= *j);
225 }

```

**Figura 28. Função: Particao**

Durante o loop, são feitas as seguintes comparações:

1. Enquanto a carta  $A[*i]$  for menor que o pivô, incrementa-se  $i$  (ou seja, avança para a próxima posição à direita);
2. Enquanto a carta  $A[*j]$  for maior que o pivô, decrementa-se  $j$  (ou seja, avança para a próxima posição à esquerda).

Quando os índices se cruzam, é verificado se  $i$  é menor ou igual a  $j$ . Se isso for verdade, troca-se as cartas nas posições  $i$  e  $j$  de lugar, atualizando os valores de  $i$  e  $j$  (incrementando  $i$  e decrementando  $j$ ). Esse processo é repetido até que  $i$  seja maior que  $j$ .

### 7.1.2. Função: Ordena

A função Ordena recebe como parâmetros o índice inicial (Esq) e final (Dir) do vetor e o array  $A$ . Ela chama a função Particao para dividir o vetor em duas partes menores. Em seguida, verifica se ainda existem elementos à esquerda ou à direita da partição e chama recursivamente a função Ordena para ordenar essas partes.

```

228 void Ordena(int Esq, int Dir, Carta A[MAX_ITEM_VETOR]){
229     int i,j;
230     Particao(Esq, Dir, &i, &j, A);
231     if (Esq < j) Ordena(Esq, j, A);
232     if (i < Dir) Ordena(i, Dir, A);
233 }

```

**Figura 29. Função: Ordena**

## 7.2. SelectionSort

O algoritmo de ordenação por seleção percorre o vetor, encontrando o menor elemento em cada iteração e o coloca na posição correta. O processo se repete até que todo o vetor esteja ordenado. O algoritmo tem uma complexidade de tempo média e pior caso de  $O(n^2)$ .

```

252 void SelectionSort(Carta A[MAX_ITEM_VETOR], int n){
253     int i, j, Min;
254     for(i=0; i<n-1; i++){
255         Min = i;
256         for(j=i+1; j<n; j++){
257             if(ComparaCartas(A[j], A[Min]) < 0){
258                 Min = j;
259             }
260         }
261         Carta aux = A[Min];
262         A[Min] = A[i];
263         A[i] = aux;
264     }
265 }

```

**Figura 30. Função: Selectionsort**

## 7.3. BubbleSort

O algoritmo utiliza dois loops aninhados para percorrer o vetor e comparar elementos adjacentes. A variável 'i' controla o número de iterações externas, enquanto a variável 'j' controla o número de iterações internas.

```

239 void BubbleSort(Carta A[MAX_ITEM_VETOR], int n){
240     int i, j;
241     for(i=0; i<n-1; i++){
242         for(j=0; j<n-i-1; j++){
243             if(ComparaCartas(A[j], A[j+1]) > 0){
244                 Carta temp = A[j];
245                 A[j] = A[j+1];
246                 A[j+1] = temp;
247             }
248         }
249     }
250 }

```

**Figura 31. Função: Selectionsort**

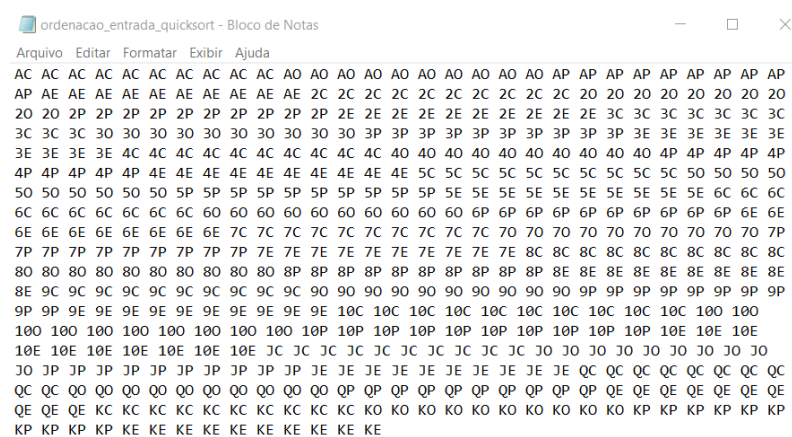
## 7.4. Resultados

```

93     if(strcmp(metodo, "quicksort") == 0){
94         QuickSort(vBaralho, quantidadeCartas(nBaralho));
95         char cartaImprimida[2];
96         for(int i=0; i<quantidadeCartas(nBaralho); i++){
97             if(vBaralho[i].valor == 1){
98                 sprintf(cartaImprimida, "A%c", vBaralho[i].naipe);
99                 fprintf(arquivo_saida, "%s ", cartaImprimida);
100             }
101             else if(vBaralho[i].valor == 11){
102                 sprintf(cartaImprimida, "J%c", vBaralho[i].naipe);
103                 fprintf(arquivo_saida, "%s ", cartaImprimida);
104             }
105             else if(vBaralho[i].valor == 12){
106                 sprintf(cartaImprimida, "Q%c", vBaralho[i].naipe);
107                 fprintf(arquivo_saida, "%s ", cartaImprimida);
108             }
109             else if(vBaralho[i].valor == 13){
110                 sprintf(cartaImprimida, "K%c", vBaralho[i].naipe);
111                 fprintf(arquivo_saida, "%s ", cartaImprimida);
112             }
113             else{
114                 sprintf(cartaImprimida, "%d%c", vBaralho[i].valor, vBaralho
[i].naipe);
115                 fprintf(arquivo_saida, "%s ", cartaImprimida);
116             }
117         }
118         fclose(arquivo_saida);
119     }

```

**Figura 32. Seleção do Quicksort**



**Figura 33. Quicksort: 11 Baralhos**

```
ordenacao_entrada_bubblesort - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
AC AO AP AE 2C 2O 2P 2E 3C 3O 3P 3E 4C 4O 4P 4E 5C 5O 5P 5E 6C 6O 6P 6E 7C 7O 7P 7E 8C
8O 8P 8E 9C 9O 9P 9E 10C 10O 10P 10E 11C 11O 11P 11E 12C 12O 12P 12E 13C 13O 13P 13E 14C 14O 14P 14E 15C 15O 15P 15E 16C 16O 16P 16E 17C 17O 17P 17E 18C 18O 18P 18E 19C 19O 19P 19E 20C 20O 20P 20E 21C 21O 21P 21E 22C 22O 22P 22E 23C 23O 23P 23E 24C 24O 24P 24E 25C 25O 25P 25E 26C 26O 26P 26E 27C 27O 27P 27E 28C 28O 28P 28E 29C 29O 29P 29E 30C 30O 30P 30E 31C 31O 31P 31E 32C 32O 32P 32E 33C 33O 33P 33E 34C 34O 34P 34E 35C 35O 35P 35E 36C 36O 36P 36E 37C 37O 37P 37E 38C 38O 38P 38E 39C 39O 39P 39E 40C 40O 40P 40E 41C 41O 41P 41E 42C 42O 42P 42E 43C 43O 43P 43E 44C 44O 44P 44E 45C 45O 45P 45E 46C 46O 46P 46E 47C 47O 47P 47E 48C 48O 48P 48E 49C 49O 49P 49E 50C 50O 50P 50E 51C 51O 51P 51E 52C 52O 52P 52E 53C 53O 53P 53E 54C 54O 54P 54E 55C 55O 55P 55E 56C 56O 56P 56E 57C 57O 57P 57E 58C 58O 58P 58E 59C 59O 59P 59E 60C 60O 60P 60E 61C 61O 61P 61E 62C 62O 62P 62E 63C 63O 63P 63E 64C 64O 64P 64E 65C 65O 65P 65E 66C 66O 66P 66E 67C 67O 67P 67E 68C 68O 68P 68E 69C 69O 69P 69E 70C 70O 70P 70E 71C 71O 71P 71E 72C 72O 72P 72E 73C 73O 73P 73E 74C 74O 74P 74E 75C 75O 75P 75E 76C 76O 76P 76E 77C 77O 77P 77E 78C 78O 78P 78E 79C 79O 79P 79E 80C 80O 80P 80E 81C 81O 81P 81E 82C 82O 82P 82E 83C 83O 83P 83E 84C 84O 84P 84E 85C 85O 85P 85E 86C 86O 86P 86E 87C 87O 87P 87E 88C 88O 88P 88E 89C 89O 89P 89E 90C 90O 90P 90E 91C 91O 91P 91E 92C 92O 92P 92E 93C 93O 93P 93E 94C 94O 94P 94E 95C 95O 95P 95E 96C 96O 96P 96E 97C 97O 97P 97E 98C 98O 98P 98E 99C 99O 99P 99E 100C 100O 100P 100E 101C 101O 101P 101E 102C 102O 102P 102E 103C 103O 103P 103E 104C 104O 104P 104E 105C 105O 105P 105E 106C 106O 106P 106E 107C 107O 107P 107E 108C 108O 108P 108E 109C 109O 109P 109E 110C 110O 110P 110E 111C 111O 111P 111E 112C 112O 112P 112E 113C 113O 113P 113E 114C 114O 114P 114E 115C 115O 115P 115E 116C 116O 116P 116E 117C 117O 117P 117E 118C 118O 118P 118E 119C 119O 119P 119E 120C 120O 120P 120E 121C 121O 121P 121E 122C 122O 122P 122E 123C 123O 123P 123E 124C 124O 124P 124E 125C 125O 125P 125E 126C 126O 126P 126E 127C 127O 127P 127E 128C 128O 128P 128E 129C 129O 129P 129E 130C 130O 130P 130E 131C 131O 131P 131E 132C 132O 132P 132E 133C 133O 133P 133E 134C 134O 134P 134E 135C 135O 135P 135E 136C 136O 136P 136E 137C 137O 137P 137E 138C 138O 138P 138E 139C 139O 139P 139E 140C 140O 140P 140E 141C 141O 141P 141E 142C 142O 142P 142E 143C 143O 143P 143E 144C 144O 144P 144E 145C 145O 145P 145E 146C 146O 146P 146E 147C 147O 147P 147E 148C 148O 148P 148E 149C 149O 149P 149E 150C 150O 150P 150E 151C 151O 151P 151E 152C 152O 152P 152E 153C 153O 153P 153E 154C 154O 154P 154E 155C 155O 155P 155E 156C 156O 156P 156E 157C 157O 157P 157E 158C 158O 158P 158E 159C 159O 159P 159E 160C 160O 160P 160E 161C 161O 161P 161E 162C 162O 162P 162E 163C 163O 163P 163E 164C 164O 164P 164E 165C 165O 165P 165E 166C 166O 166P 166E 167C 167O 167P 167E 168C 168O 168P 168E 169C 169O 169P 169E 170C 170O 170P 170E 171C 171O 171P 171E 172C 172O 172P 172E 173C 173O 173P 173E 174C 174O 174P 174E 175C 175O 175P 175E 176C 176O 176P 176E 177C 177O 177P 177E 178C 178O 178P 178E 179C 179O 179P 179E 180C 180O 180P 180E 181C 181O 181P 181E 182C 182O 182P 182E 183C 183O 183P 183E 184C 184O 184P 184E 185C 185O 185P 185E 186C 186O 186P 186E 187C 187O 187P 187E 188C 188O 188P 188E 189C 189O 189P 189E 190C 190O 190P 190E 191C 191O 191P 191E 192C 192O 192P 192E 193C 193O 193P 193E 194C 194O 194P 194E 195C 195O 195P 195E 196C 196O 196P 196E 197C 197O 197P 197E 198C 198O 198P 198E 199C 199O 199P 199E 200C 200O 200P 200E 201C 201O 201P 201E 202C 202O 202P 202E 203C 203O 203P 203E 204C 204O 204P 204E 205C 205O 205P 205E 206C 206O 206P 206E 207C 207O 207P 207E 208C 208O 208P 208E 209C 209O 209P 209E 210C 210O 210P 210E 211C 211O 211P 211E 212C 212O 212P 212E 213C 213O 213P 213E 214C 214O 214P 214E 215C 215O 215P 215E 216C 216O 216P 216E 217C 217O 217P 217E 218C 218O 218P 218E 219C 219O 219P 219E 220C 220O 220P 220E 221C 221O 221P 221E 222C 222O 222P 222E 223C 223O 223P 223E 224C 224O 224P 224E 225C 225O 225P 225E 226C 226O 226P 226E 227C 227O 227P 227E 228C 228O 228P 228E 229C 229O 229P 229E 230C 230O 230P 230E 231C 231O 231P 231E 232C 232O 232P 232E 233C 233O 233P 233E 234C 234O 234P 234E 235C 235O 235P 235E 236C 236O 236P 236E 237C 237O 237P 237E 238C 238O 238P 238E 239C 239O 239P 239E 240C 240O 240P 240E 241C 241O 241P 241E 242C 242O 242P 242E 243C 243O 243P 243E 244C 244O 244P 244E 245C 245O 245P 245E 246C 246O 246P 246E 247C 247O 247P 247E 248C 248O 248P 248E 249C 249O 249P 249E 250C 250O 250P 250E 251C 251O 251P 251E 252C 252O 252P 252E 253C 253O 253P 253E 254C 254O 254P 254E 255C 255O 255P 255E 256C 256O 256P 256E 257C 257O 257P 257E 258C 258O 258P 258E 259C 259O 259P 259E 260C 260O 260P 260E 261C 261O 261P 261E 262C 262O 262P 262E 263C 263O 263P 263E 264C 264O 264P 264E 265C 265O 265P 265E 266C 266O 266P 266E 267C 267O 267P 267E 268C 268O 268P 268E 269C 269O 269P 269E 270C 270O 270P 270E 271C 271O 271P 271E 272C 272O 272P 272E 273C 273O 273P 273E 274C 274O 274P 274E 275C 275O 275P 275E 276C 276O 276P 276E 277C 277O 277P 277E 278C 278O 278P 278E 279C 279O 279P 279E 280C 280O 280P 280E 281C 281O 281P 281E 282C 282O 282P 282E 283C 283O 283P 283E 284C 284O 284P 284E 285C 285O 285P 285E 286C 286O 286P 286E 287C 287O 287P 287E 288C 288O 288P 288E 289C 289O 289P 289E 290C 290O 290P 290E 291C 291O 291P 291E 292C 292O 292P 292E 293C 293O 293P 293E 294C 294O 294P 294E 295C 295O 295P 295E 296C 296O 296P 296E 297C 297O 297P 297E 298C 298O 298P 298E 299C 299O 299P 299E 300C 300O 300P 300E 301C 301O 301P 301E 302C 302O 302P 302E 303C 303O 303P 303E 304C 304O 304P 304E 305C 305O 305P 305E 306C 306O 306P 306E 307C 307O 307P 307E 308C 308O 308P 308E 309C 309O 309P 309E 310C 310O 310P 310E 311C 311O 311P 311E 312C 312O 312P 312E 313C 313O 313P 313E 314C 314O 314P 314E 315C 315O 315P 315E 316C 316O 316P 316E 317C 317O 317P 317E 318C 318O 318P 318E 319C 319O 319P 319E 320C 320O 320P 320E 321C 321O 321P 321E 322C 322O 322P 322E 323C 323O 323P 323E 324C 324O 324P 324E 325C 325O 325P 325E 326C 326O 326P 326E 327C 327O 327P 327E 328C 328O 328P 328E 329C 329O 329P 329E 330C 330O 330P 330E 331C 331O 331P 331E 332C 332O 332P 332E 333C 333O 333P 333E 334C 334O 334P 334E 335C 335O 335P 335E 336C 336O 336P 336E 337C 337O 337P 337E 338C 338O 338P 338E 339C 339O 339P 339E 340C 340O 340P 340E 341C 341O 341P 341E 342C 342O 342P 342E 343C 343O 343P 343E 344C 344O 344P 344E 345C 345O 345P 345E 346C 346O 346P 346E 347C 347O 347P 347E 348C 348O 348P 348E 349C 349O 349P 349E 350C 350O 350P 350E 351C 351O 351P 351E 352C 352O 352P 352E 353C 353O 353P 353E 354C 354O 354P 354E 355C 355O 355P 355E 356C 356O 356P 356E 357C 357O 357P 357E 358C 358O 358P 358E 359C 359O 359P 359E 360C 360O 360P 360E 361C 361O 361P 361E 362C 362O 362P 362E 363C 363O 363P 363E 364C 364O 364P 364E 365C 365O 365P 365E 366C 366O 366P 366E 367C 367O 367P 367E 368C 368O 368P 368E 369C 369O 369P 369E 370C 370O 370P 370E 371C 371O 371P 371E 372C 372O 372P 372E 373C 373O 373P 373E 374C 374O 374P 374E 375C 375O 375P 375E 376C 376O 376P 376E 377C 377O 377P 377E 378C 378O 378P 378E 379C 379O 379P 379E 380C 380O 380P 380E 381C 381O 381P 381E 382C 382O 382P 382E 383C 383O 383P 383E 384C 384O 384P 384E 385C 385O 385P 385E 386C 386O 386P 386E 387C 387O 387P 387E 388C 388O 388P 388E 389C 389O 389P 389E 390C 390O 390P 390E 391C 391O 391P 391E 392C 392O 392P 392E 393C 393O 393P 393E 394C 394O 394P 394E 395C 395O 395P 395E 396C 396O 396P 396E 397C 397O 397P 397E 398C 398O 398P 398E 399C 399O 399P 399E 400C 400O 400P 400E 401C 401O 401P 401E 402C 402O 402P 402E 403C 403O 403P 403E 404C 404O 404P 404E 405C 405O 405P 405E 406C 406O 406P 406E 407C 407O 407P 407E 408C 408O 408P 408E 409C 409O 409P 409E 410C 410O 410P 410E 411C 411O 411P 411E 412C 412O 412P 412E 413C 413O 413P 413E 414C 414O 414P 414E 415C 415O 415P 415E 416C 416O 416P 416E 417C 417O 417P 417E 418C 418O 418P 418E 419C 419O 419P 419E 420C 420O 420P 420E 421C 421O 421P 421E 422C 422O 422P 422E 423C 423O 423P 423E 424C 424O 424P 424E 425C 425O 425P 425E 426C 426O 426P 426E 427C 427O 427P 427E 428C 428O 428P 428E 429C 429O 429P 429E 430C 430O 430P 430E 431C 431O 431P 431E 432C 432O 432P 432E 433C 433O 433P 433E 434C 434O 434P 434E 435C 435O 435P 435E 436C 436O 436P 436E 437C 437O 437P 437E 438C 438O 438P 438E 439C 439O 439P 439E 440C 440O 440P 440E 441C 441O 441P 441E 442C 442O 442P 442E 443C 443O 443P 443E 444C 444O 444P 444E 445C 445O 445P 445E 446C 446O 446P 446E 447C 447O 447P 447E 448C 448O 448P 448E 449C 449O 449P 449E 450C 450O 450P 450E 451C 451O 451P 451E 452C 452O 452P 452E 453C 453O 453P 453E 454C 454O 454P 454E 455C 455O 455P 455E 456C 456O 456P 456E 457C 457O 457P 457E 458C 458O 458P 458E 459C 459O 459P 459E 460C 460O 460P 460E 461C 461O 461P 461E 462C 462O 462P 462E 463C 463O 463P 463E 464C 464O 464P 464E 465C 465O 465P 465E 466C 466O 466P 466E 467C 467O 467P 467E 468C 468O 468P 468E 469C 469O 469P 469E 470C 470O 470P 470E 471C 471O 471P 471E 472C 472O 472P 472E 473C 473O 473P 473E 474C 474O 474P 474E 475C 475O 475P 475E 476C 476O 476P 476E 477C 477O 477P 477E 478C 478O 478P 478E 479C 479O 479P 479E 480C 480O 480P 480E 481C 481O 481P 481E 482C 482O 482P 482E 483C 483O 483P 483E 484C 484O 484P 484E 485C 485O 485P 485E 486C 486O 486P 486E 487C 487O 487P 487E 488C 488O 488P 488E 489C 489O 489P 489E 490C 490O 490P 490E 491C 491O 491P 491E 492C 492O 492P 492E 493C 493O 493P 493E 494C 494O 494P 494E 495C 495O 495P 495E 496C 496O 496P 496E 497C 497O 497P 497E 498C 498O 498P 498E 499C 499O 499P 499E 500C 500O 500P 500E 501C 501O 501P 501E 502C 502O 502P 502E 503C 503O 503P 503E 504C 504O 504P 504E 505C 505O 505P 505E 506C 506O 506P 506E 507C 507O 507P 507E 508C 508O 508P 508E 509C 509O 509P 509E 510C 510O 510P 510E 511C 511O 511P 511E 512C 512O 512P 512E 513C 513O 513P 513E 514C 514O 514P 514E 515C 515O 515P 515E 516C 516O 516P 516E 517C 517O 517P 517E 518C 518O 518P 518E 519C 519O 519P 519E 520C 520O 520P 520E 521C 521O 521P 521E 522C 522O 522P 522E 523C 523O 523P 523E 524C 524O 524P 524E 525C 525O 525P 525E 526C 526O 526P 526E 527C 527O 527P 527E 528C 528O 528P 528E 529C 529O 529P 529E 530C 530O 530P 530E 531C 531O 531P 531E 532C 532O 532P 532E 533C 533O 533P 533E 534C 534O 534P 534E 535C 535O 535P 535E 536C 536O 536P 536E 537C 537O 537P 537E 538C 538O 538P 538E 539C 539O 539P 539E 540C 540O 540P 540E 541C 541O 541P 541E 542C 542O 542P 542E 543C 543O 543P 543E 544C 544O 544P 544E 545C 545O 545P 545E 546C 546O 546P 546E 547C 547O 547P 547E 548C 548O 548P 548E 549C 549O 549P 549E 550C 550O 550P 550E 551C 551O 551P 551E 552C 552O 552P 552E 553C 553O 553P 553E 554C 554O 554P 554E 555C 555O 555P 555E 556C 556O 556P 556E 557C 557O 557P 557E 558C 558O 558P 558E 559C 559O 559P 559E 560C 560O 560P 560E 561C 561O 561P 561E 562C 562O 562P 562E 563C 563O 563P 563E 564C 564O 564P 564E 565C 565O 565P 565E 566C 566O 566P 566E 567C 567O 567P 567E 568C 568O 568P 568E 569C 569O 569P 569E 570C 570O 570P 570E 571C 571O 571P 571E 572C 572O 572P 572E 573C 573O 573P 573E 574C 574O 574P 574E 575C 575O 575P 575E 576C 576O 576P 576E 577C 577O 577P 577E 578C 578O 578P 578E 579C 579O 579P 579E 580C 580O 580P 580E 581C 581O 581P 581E 582C 582O 582P 582E 583C 583O 583P 583E 584C 584O 584P 584E 585C 585O 585P 585E 586C 586O 586P 586E 587C 587O 587P 587E 588C 588O 588P 588E 589C 589O 589P 589E 590C 590O 590P 590E 591C 591O 591P 591E 592C 592O 592P 592E 593C 593O 593P 593E 594C 594O 594P 594E 595C 595O 595P 595E 596C 596O 596P 596E 597C 597O 597P 597E 598C 598O 598P 598E 599C 599O 599P 599E 600C 600O 600P 600E 601C 601O 601P 601E 602C 602O 602P 602E 603C 603O 603P 603E 604C 604O 604P 604E 605C 605O 605P 605E 606C 606O 606P 606E 607C 607O 607P 607E 608C 608O 608P 608E 609C 609O 609P 609E 610C 610O 610P 610E 611C 611O 611P 611E 612C 612O 612P 612E 613C 613O 613P 613E 614C 614O 614P 614E 615C 615O 615P 615E 616C 616O 616P 616E 617C 617O 617P 617E 618C 618O 618P 618E 619C 619O 619P 619E 620C 620O 620P 620E 621C 621O 621P 621E 622C 622O 622P 622E 623C 623O 623P 623E 624C 624O 624P 624E 625C 625O 625P 625E 626C 626O 626P 626E 627C 627O 627P 627E 628C 628O 628P 628E 629C 629O 629P 629E 630C 630O 630P 630E 631C 631O 631P 631E 632C 632O 632P 632E 633C 633O 633P 633E 634C 634O 634P 634E 635C 635O 635P 635E 636C 636O 636P 636E 637C 637O 637P 637E 638C 638O 638P 638E 639C 639
```