Alinea A

Caros estudantes,

Tenho uma lista de excertos de código e respetivo comentário. Esta é a primeira alínea, supostamente simples mas houve muita complicação. Consultar erros de qualidade (EQx) na página 170.

Exemplo 1 - EQ9

```
  if (valor==0) printf("AP");

  if (valor==1) printf("2P");

  ...

  if (valor==51) printf("RE");
```

Este método exaustivo, para cada entrada possível colocar a saída esperada, é a pior utilização possível das potencialidades da programação. Neste caso o número de dados de entrada válidos é reduzido, muito raramente tal ocorre, mas em todo o caso o programa pode ser muito mais reduzido que a enumeração exaustiva.

Exemplo 2 - EQ3

```
char paus='P';

char ouros='O';

char copas='C';

char espadas='E';
```

Falha clara aqui de um vetor, para agregar variáveis idênticas. Esta opção de estrutura de dados, inviabiliza mais tarde a eliminação de código duplicado (neste caso quadriplicado), pelo que o erro EQ9 acaba por ser também aplicável.

Exemplo 3 - EQ4

```
 if (valor == 0) {

   char b[10] = "AP";

   printf("%s",b);

 }

 ...

 if (valor == 51) {

   char b[10] =  "RE";

   printf("%s\n",b);
```

```
 }
```

Este é também o método exaustivo, mas com a utilização de uma variável temporária local, que para além de ser desnecessária dado aplicar-se o EQ28 uma atribuição uma leitura, mais importante é o nome sem significado, com apenas uma letra.

Exemplo 4 - sem erro de qualidade

```
char *cartas[52]={"AP","2P","3P","4P","5P","6P","7P","8P","9P","10P","VP","DP
","RP","AO","2O","3O","4O","5O"," 6O","7O","8O","9O","10O","VO","DO","RO","AC
","2C","3C","4C","5C","6C","7C","8C","9C","10 C","VC","DC","RC","AE","2E","3E
","4E","5E","6E","7E","8E","9E","10E","VE","DE","RE"};
```

Não se aplica erro de qualidade a esta opção. No entanto este código não deve ser feito, porque ignora o facto de poder construir os dados no vetor de forma mais simples, com base em dois vetores, um com os números e outro com os naipes. Se a informação pode ser obtida por uma expressão, não há necessidade nem deve ser colocada por extenso. Compreende-se aqui que o programador queira um acesso imediato à carta.

Exemplo 5 - sem erro de qualidade

```
for(i=0;i<52;i++)

  if(baralho[valor]==baralho[i])

    printf("%s\n",baralho[i]);
```

Exemplo de código ineficiente, utilizando o ciclo for para localizar um valor que já é conhecido. Simplesmente printf("%s\n",baralho[valor]); seria o suficiente.

Exemplo 6 - Estrutura de dados aconselhada e algoritmo

```
char valor[]={'A','2','3','4','5','6','7','8','9','0','V','D','R'};

char naipe[]={'P','O','C','E'};

...

if (valor % 13 == 9)

  printf("1");

printf("%c%c", numero[valor % 13], naipe[valor / 13]);
```

Esta é a opção natural. Há no entanto formas mais compactas: char *valor="A234567891VDR"; char *naipe="POCE"; Convém identificar os padrões que se repetem, de modo a colocar no código o que é realmente diferente. O que é igual, pode-se processar de uma só vez. Vamos supor que querem agora o jogo um baralho internacional, as figuras mudam de letra e os naipes. Em vez de alterar 52 cartas alteram apenas o vetor de naipes e números. Se quiserem um baralho com mais números e naipes, também não há problema.

Exemplo 7 - sem erro de qualidade

```
char *cartas[] = {"A","2","3","4","5","6","7","8","9","10","V","D","R"};

char *naipes[] = {"P","O","C","E"};
```

Por causa de um elemento, em vez de guardar um caracter guarda uma string. É uma opção válida, mas não aconselho. Podem fazer uma função para se poderem abstrair do caso especial do 10. Estamos a falar de bytes, um só bloco de memória, pelo que neste caso tanto faz, mas fica aqui a nota.

Exemplo 8 - sem erro de qualidade

```
do {

...

  contador++;

  valor=valor-13;

} while(valor>=0);
```

Temos um ciclo quando podemos ter uma expressão, basta contador=valor/13; e o último valor utilizável seria valor%=13;

Exemplo 9 - sem erro de qualidade

```
 int numero[13]={65,2,3,4,5,6,7,8,9,10,86,68,82};
```

No mesmo vetor são colocados números dos caracteres em ASCII, e nesse caso bastaria utilizar as constantes 'A' para o 65, já que é essa a utilização dada. Mas é utilizado também no vetor números com significado de número, como o 2. Normalmente não é aconselhado colocar no mesmo vetor itens com naturezas/significados distintos. Deveria criar dois vetores, ou utilizado o '2' em vez do 2. É claro uma forma de resolver a questão do 10, que tem dois caracteres, pelo que o código fica com a mesma dimensão.

Exemplo 10 - EQ12

```
switch(i_carta) {

  case 0:

  printf("%c",'A');

  break;

  ...

  case 10:

  printf("%c",'V');

  break;

  ...
```

Esta situação de utilização para um índice, é clara a falha da utilização de um vetor com constantes. É visível aqui as instruções parecidas em cada caso do switch, pelo que esta situação é facilmente identificável.

Exemplo 11 - sem erro de qualidade

```
 char naipes[4]="POCE";
```

Atenção que uma string de 4 caracteres necessita de 5 bytes. A declaração tem de omitir o tamanho, ou então colocar o tamanho correto que é 5.

Exemplo 12 - EQ3

```
char *P[13] = { "AP", "2P" , "3P","4P","5P","6P","7P","8P","9P","10P","VP","DP","RP" };

char *O[13] = { "AO", "2O" , "3O","4O","5O","6O","7O","8O","9O","10O","VO","DO","RO" };

char *C[13] = { "AC", "2C" , "3C","4C","5C","6C","7C","8C","9C","10C","VC","DC","RC" };

char *E[13] = { "AE", "2E" , "3E","4E","5E","6E","7E","8E","9E","10E","VE","DE","RE" };
```

Este exemplo coloca uma variável por cada naipe. Embora possa ser atribuído o EQ4, dado que existem nomes com uma só letra, o EQ3 é mais focado ao problema existente. Qualquer código que utilize estes vetores, terá de ser quadriplicado por cada naipe, quando até os naipes não são relevantes para este jogo.

Exemplo 13 - sem erro de qualidade

```
char * Carta(int carta) {

  static char resultado[4], *naipe="POCE", *numero="A234567890VDR";

  sprintf(resultado,"1%c%c", numero[valor % 13], naipe[valor / 13]);

  if (valor % 13 == 9)

    return resultado;

  return resultado+1;

}
```

Imprime uma carta para string, que retorna, de modo a ser utilizado de forma genérica (exemplo utilizado por um estudante apenas, mas é apresentada uma versão ligeiramente alterada). É uma excelente opção, mas atenção que como utiliza uma string estática que retorna, não resulta o seguinte: printf("%s %s", Carta(23), Carta(10)); Seria impressa apenas a carta 10 duas vezes, e não a carta 23 e 10. É exactamente por este motivo que o strtok não funciona de forma aninhada. No caso do e-fólio, como a única utilização para o nome das cartas é imprimir, considero mais razoável uma função que imprima logo, e pode ter nome sugestivo que imprime a carta, tipo MostraCarta(int carta).

Exemplo 14 - EQ28

```
char getNaipe(int valor) {

  char naipes[4] = { 'P','O','C','E' };

  int naipeIdx = valor / 13;

  char naipe = naipes[naipeIdx];

  return naipe;

}
```

Aqui aplica-se o EQ28, uma atribuição uma utilização, que seria facilmente removido, mas o mais importante aqui é a abstração funcional. Quem faz esta faz também a do número. Embora aceitável, não se justifica dado que o texto da carta é utilizado apenas para mostrar a carta para o utilizador, não tem mais utilidade.

Se existir alguma outra situação que pretendam comentado e não listado aqui, coloquem.

Cumprimentos,
José Coelho

Alinea B

Caros estudantes,

Seguem agora os exemplos relativos à alínea B, na qual poderiam utilizar a AF baralhar.c, coloquei essa indicação expressa no enunciado. Aqui o maior problema foi não terem identificado o tipo inteiro para representar uma carta. A alínea A pretendia direccionar a vossa opção para esta representação, é por esse motivo que a um número faz corresponder uma carta. Mas este ponto será mais claro na alínea C. Novamente podem ver os erros de qualidade na página 170.

Exemplo 1 - EQ9

```
char *baralho[208]={"AP","2P","3P","4P","5P","6P","7P","8P","9P","10P","DP","
VP","RP",                          "AO","2O","3O","4O","5O","6O","7O","8O","9O"
,"10O","DO","VO","RO",

         "AC","2C","3C","4C","5C","6C","7C","8C","9C","10C","DC","VC","RC",

         "AE","2E","3E","4E","5E","6E","7E","8E","9E","10E","DE","VE","RE",

         "AP","2P","3P","4P","5P","6P","7P","8P","9P","10P","DP","VP","RP",

         "AO","2O","3O","4O","5O","6O","7O","8O","9O","10O","DO","VO","RO",

         "AC","2C","3C","4C","5C","6C","7C","8C","9C","10C","DC","VC","RC",

         "AE","2E","3E","4E","5E","6E","7E","8E","9E","10E","DE","VE","RE",
```

```
        "AP","2P","3P","4P","5P","6P","7P","8P","9P","10P","DP","VP","RP",
        "AO","2O","3O","4O","5O","6O","7O","8O","9O","10O","DO","VO","RO",
        "AC","2C","3C","4C","5C","6C","7C","8C","9C","10C","DC","VC","RC",
        "AE","2E","3E","4E","5E","6E","7E","8E","9E","10E","DE","VE","RE",
        "AP","2P","3P","4P","5P","6P","7P","8P","9P","10P","DP","VP","RP",
        "AO","2O","3O","4O","5O","6O","7O","8O","9O","10O","DO","VO","RO",
        "AC","2C","3C","4C","5C","6C","7C","8C","9C","10C","DC","VC","RC",
        "AE","2E","3E","4E","5E","6E","7E","8E","9E","10E","DE","VE","RE"};
```

Se as 52 cartas por extenso está no limite, aqui já é demais. Tem os baralhos duplicados, o que provoca constantes duplicadas, sendo este código desaconselhado. Pode indiciar falta de prática em ciclos, bem como em expressões, só assim se justifica a tentação de colocar num vetor todas as cartas por extenso.

Exemplo 2 - EQ3

```
char *b1[52]={"AP" , "2P" , "3P" , "4P" , "5P" , "6P" , "7P" , "8P" , "9P" ,
"10P" , "VP" , "DP" , "RP" ,
             "AO" , "2O" , "3O" , "4O" , "5O" , "6O" , "7O" , "8O" , "9O" ,
"10O" , "VO" , "DO" , "RO" ,
             "AC" , "2C" , "3C" , "4C" , "5C" , "6C" , "7C" , "8C" , "9C" ,
"10C" , "VC" , "DC" , "RC" ,
             "AE" , "2E" , "3E" , "4E" , "5E" , "6E" , "7E" , "8E" , "9E" ,
"10E" , "VE" , "DE" , "RE"};
char *b2[52]={"AP" , "2P" , "3P" , "4P" , "5P" , "6P" , "7P" , "8P" , "9P" ,
"10P" , "VP" , "DP" , "RP" ,
             "AO" , "2O" , "3O" , "4O" , "5O" , "6O" , "7O" , "8O" , "9O" ,
"10O" , "VO" , "DO" , "RO" ,
             "AC" , "2C" , "3C" , "4C" , "5C" , "6C" , "7C" , "8C" , "9C" ,
"10C" , "VC" , "DC" , "RC" ,
             "AE" , "2E" , "3E" , "4E" , "5E" , "6E" , "7E" , "8E" , "9E" ,
"10E" , "VE" , "DE" , "RE"};
...
```

Este é o mesmo problema de colocar uma variável para cada naipe, mas agravado por ser em matrizes iguais, o EQ9 aplica-se também ou no código que utilizar estas variáveis.

Exemplo 3 - EQ9

```
cartasvetor[208]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
23,24,25,26,2 7,28,29,30,31,32,33,34,35,

                  36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55
,56,57,58,59,60,61,62,63,64,65 ,66,67,

                  68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87
,88,89,90,91,92,93,94,95,96,97 ,98,99,

                  100,101,102,103,104,105,106,107,108,109,110,111,112,113,114
,115,116,117,118,119,120, 121,122,123,

                  124,125,126,127,128,129,130,131,132,133,134,135,136,137,138
,139,140,141,142,143,144, 145,146,147,

                  148,149,150,151,152,153,154,155,156,157,158,159,160,161,162
,163,164,165,166,167,168, 169,170,171,

                  172,173,174,175,176,177,178,179,180,181,182,183,184,185,186
,187,188,189,190,191,192, 193,194,195,

                  196,197,198,199,200,201,202,203,204,205,206,207}
```

Junção dos 4 baralhos fazendo cartas diferentes, em vez de serem os números de 0 a 51. Naturalmente que não só esta situação acaba por distinguir entidades distintas quando se querem iguais (a mesma carta de diferentes baralhos), como provoca no código que utiliza esta estrutura de dados o erro EQ9.

Exemplo 4 - Abstração aconselhada

```
void Baralhar(int vetor[], int n)
```

Esta função existia numa atividade formativa. Poderiam até ter reutilizado a implementação de um colega, mas era uma abstração natural, já que a função baralhar é independente dos dados e assim pode ser reutilizada mais tarde, quando têm 4 baralhos em vez de 1.

Exemplo 5 - Abstração aconselhada

```
void MostrarCartas(int vetor[], int n)
```

Esta função poderia ser utilizada mais tarde, ou então uma função no singular, para mostrar uma só carta. Assume um ponto muito importante, que é considerarem uma carta um inteiro. Mais sobre este assunto na alínea C.

Exemplo 6 - sem erro de qualidade

```
  for(j=0; j<=207; j++) {

    while(i-->0)

      randaux();

  ...
```

O ciclo while pode estar dentro de outro ciclo, já que após a variável i ficar a zero, pode ser chamada várias vezes que não tem efeito. No entanto este ciclo while nada diz respeito a

outro ciclo for. Cada instrução deve estar no local que lhe diga respeito, dentro de um ciclo devem estar apenas as instruções relativas ao que se pretende fazer com o ciclo, e não outras. Neste caso o ciclo while deveria ser chamado antes do ciclo for.

Exemplo 7 - sem erro de qualidade

```
char aux[4];

for (int j = 0; j < 4; j++)

  aux[j]=para[j];
```

Cópia de strings. Existe a função strcpy.

Exemplo 8 - sem erro de qualidade

```
scanf("%d", &i);

while (i-->=0)    {

  randaux();

  baralhar(v_baralho, 208);

...
```

Este foi um erro cometido por alguns estudantes. O ciclo fornecido deveria apenas chamar a função randaux um número i de vezes, nada mais. Foram colocadas outras instruções dentro do ciclo, ficando naturalmente a funcionalidade inicial alterada.

Exemplo 9 - sem erro de qualidade

```
for (i=0; v_baralho[i]!=v_baralho[10]; i++)
```

Este ciclo pretende imprimir as 10 primeiras cartas, então pára quando a carta for igual à que está após a última. Como há cartas iguais, pode acontecer que o teste de paragem não seja após a décima carta. Para quê esta expressão, não será mais simples i<10, que funciona sempre independentemente dos dados de entrada?

Exemplo 10 - sem erro de qualidade

```
  for(i=1;i<=NUMCARTAS;i++)

    baralhos[i]=i;
```

Inicializa os 4 baralhos distinguindo as mesmas cartas entre baralhos. Falta aqui agregar as cartas dos diferentes baralhos são iguais, pelo que não devem ser distinguidas de forma artificial.

Exemplo 11 - sem erro de qualidade

```
strcpy (carta, baralho[R+n]);

strcpy (baralho[R+n], baralho[n]);
```

```
strcpy (baralho[n], carta);
```

Baralha as próprias strings das cartas, em vez de baralhar simplesmente um vetor com índices das cartas. Provoca cópia de strings em vez de simples atribuições entre números. De modo geral, não há justificação para se baralhar ou ordenar os elementos, apenas índices, de modo a poupar na operação de troca de elementos.

Exemplo 12 - sem erro de qualidade

```
char cartas[16][13][2]; // 4 baralhos

char cartas[4][13][2];  // 1 baralho
```

Nesta estrutura de dados, utilizada por um só estudante, para guardar um baralho (ou 4), foram utilizadas 3 dimensões. No entanto a divisão não é por naipes, já que no final de cartas[0][0] pode estar qualquer carta, e não um Ás do primeiro naipe. Este é um exemplo de adição de uma complexidade na entidade cartas, que é a sua disposição em conjuntos de 13, que não existe à partida. O baralho é o conjunto de todas as cartas, pelo que deve existir um índice para aceder a qualquer carta. Neste caso é necessário dois índices. Todo o código fica mais complexo devido a esta decisão na estrutura de dados.

Cumprimentos,
José Coelho

Alinea C

Caros estudantes,

Segue-se os exemplos da alínea C.

Exemplo 1 - EQ9

```
 if(baralhos[i]==0 || baralhos[i]==13 || baralhos[i]==26 || baralhos[i]==39)
```

Estas expressões por extenso, quando bastaria if(baralhos[i]%13==0), duplicam código na própria expressão, e neste caso podem ser substituídas por uma expressão menor. Se não fosse possível escrever uma expessão menor, então deveriam criar uma função, mas as operações de divisão inteira e resto de divisão, têm de as saber utilizar. São operações básicas, executadas diretamente no computador, não faz sentido expandir essas expressões apenas para não utilizar o resto da divisão inteira.

Exemplo 2 - EQ4, EQ12

```
switch(a)    {

  case 'A': return 11;

  case '2': return 2;

  case '3': return 3;

  case '4': return 4;
```

```
  ...
```

Este caso o switch pode ser substituído por uma expressão, o strchr por exemplo, identifica as figuras e o 10, teste com 'A' e o resto sendo dígito por diferença de código com '0'. Aqui a própria letra para identificar a variável, gera também o EQ4.

Exemplo 3 - EQ9

```
 if ((baralho[i]=="2P") || (baralho[i]=="2O") || (baralho[i]=="2C") || (baral
ho[i]=="2E")){
```

Para além do erro de qualidade, na própria expressão que está repetida, mas também nos outros condicionais para o resto dos valores, existe algo que não devem fazer aqui. Estão a comparar strings. Se as strings resultarem de constantes, como o compilador optimiza e coloca todas as strings constantes no mesmo espaço de memória, o código até é capaz de funcionar. Mas qualquer string que exista criada em tempo de execução, irá falhar. Devem utilizar o strcmp para comparar strings.

Exemplo 4 - EQ9

```
   int pontos[52] = {        11, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10,

                             11, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10,

                             11, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10,

                             11, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10,

   };
```

Seria aceitável embora seja preferível uma expressão, a utilização da pontuação com base no número da carta. Mas ter a informação duplicada para cada naipe, tem de se aplicar o EQ9. Bastaria utilizar o vetor pontos[carta%13] para obter o mesmo que se conseguiria com pontos[carta].

Exemplo 5 - EQ9

```
       if(baralhado[x]=='AP'||'AO'||'AC'||'AE'){
```

Nesta expressão há várias questões. Primeiro as plicas são para letras isoladas, e está a colocar-se uma string (duas letras). Segundo, existem na verdade 4 expressões lógicas, das quais a variável baralhado[x] é utilizada em apenas uma. As restantes são todas positivas. Se se quer uma expressão para saber se uma variável X tem um valor de um conjunto de constantes, por exemplo 1, 4 e 7, tem de se colocar por extenso todas as possibilidades: X==1||X==4||X==7. Fazendo X==1||4||7 o resultado será sempre verdadeiro, já que tanto o 4 como o 7 são diferentes de 0 e portanto são expressões verdadeiras. Após o erro corrigido, aplicaria-se o erro EQ9 já que a própria expressão tem código duplicado, para além de que os condicionais seguintes também

Exemplo 6 - EQ20

```
valor[208]={11,2,3,4,5,6,7,8,9,10,10,10,10,11,2,3,4,5,6,7,8,9,10,10,10,10,11,
2,3,4,5,6,7,8,9,1 0,10,10,10,11,2,3,4,5,6,7,8,9,10,10,10,10,11,2,3,4,5,6,7,8,
9,10,10,10,10,11,2,3,4,5,6,7,8,9,10, 10,10,10,11,2,3,4,5,6,7,8,9,10,10,10,10,
11,2,3,4,5,6,7,8,9,10,10,10,10,11,2,3,4,5,6,7,8,9,10,1 0,10,10,11,2,3,4,5,6,7
```

```
,8,9,10,10,10,10,11,2,3,4,5,6,7,8,9,10,10,10,10,11,2,3,4,5,6,7,8,9,10,10, 10,
10,11,2,3,4,5,6,7,8,9,10,10,10,10,11,2,3,4,5,6,7,8,9,10,10,10,10,11,2,3,4,5,6
,7,8,9,10,10,1 0,10,11,2,3,4,5,6,7,8,9,10,10,10,10};
```

Para além do vetor com os 4 baralhos, é criado um vetor paralelo com a pontuação de cada carta. Atendendo a que esta informação pode ser obtido através da carta, estes dados são considerados desnecessários (EQ20).

Exemplo 7 - sem erro de qualidade

```
 while(pontos<=21 && pontos<17)
```

Esta expressão lógica é uma concatenação de duas expressões lógicas, em que uma delas é sempre verdadeira quando a outra é verdadeira. Basta portanto manter apenas a expressão mais limitadora, neste caso pontos<17

Exemplo 8 - EQ12

```
  switch(valorcarta)    {
    case 'V':
    case 'D':
    case 'R':
    case '1':
       return 10;
    case '2':
       return 2;
  ...
```

Exemplo da enumeração de todas as possibilidades. Não faz sentido enumerações quando se pode utilizar uma expressão. É o mesmo que em vez de fazer x=x*2 colocar um switch com todas as possibilidades.

Exemplo 9 - EQ9

```
 ...
         case 2:
           printf("%s %s : %d pontos.",vetor[i],vetor[i+1],soma);
           break;
         case 3:
           printf("%s %s %s : %d pontos.",vetor[i],vetor[i+1],vetor[i+2],soma
);
           break; …
```

Este erro seria também o EQ12. Mas aqui é mesmo o EQ9, algo que poderia ser colocado de forma genérica com um ciclo, é colocada de forma enumerada. Se existirem k cartas, imprime k números e a pontuação. É um exemplo claro de falha de um ciclo, algo a refletir o porquê desta solução ter surgido, em vez de percorrer os elementos que existiam.

Exemplo 10 - Estrutura de dados aconselhada para carta

```
int carta; // valor da carta um inteiro.
```

Inteiro com o número da carta, em vez de strings para identificar cartas. A alínea A solicita um número e devolve uma carta. Esperava que pudessem ter compreendido que apenas para mostrar é que é necessário strings, tudo o resto uma carta pode ser um inteiro. Na alínea B a função Baralhar seria com inteiros e não com strings.

Exemplo 11 - Abstracção aconselhada

```
DarCarta( argumentos conforme a estrutura de dados ) // dá uma carta e atuali
za pontos.

// Por exemplo:

if (carta % 13 == 0) {

  contadorAs++;

  pontos += 10;

}

if (carta % 13 >= 9)

  pontos += 10;

else

  pontos += 1 + carta % 13;

if (pontos > 21 && contadorAs > 0) {

  pontos -= 10;

  contadorAs--;

}

// alternativa a solução de vetor para os pontos de uma carta é também boa da
do que há apenas 13 números:

int *pontos = { 11, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10};
```

Esta função é importante para poder ser reutilizada tanto pela banca como pelos jogadores. Convém distinguir a carta e contabilização de pontos, da estratégia, decisão quando parar. Podem ver no código para contabilização de pontos, algo que inclui as três situações, o Ás (mais complexo), o número que corresponde aos pontos, e a figura que valem 10. Nada de grandes complicações. A estrutura de dados auxiliar com um vetor com a pontuação seria também uma boa opção, dado que são poucos números.

Cumprimentos,
José Coelho

Alinea D

Caros estudantes,

Segue por fim a alínea D, com menos exemplos. Esta alínea estará mais mal tratada, pelo que coloquem aqui situações que pretendam ver analisadas.

Exemplo 1 - EQ3

```
char  jogador1[MAO][TAM], jogador2[MAO][TAM], jogador3[MAO][TAM], jogador4[MA
O][TAM];
```

Exemplo claro deste tipo de erro, cometido por vários estudantes. Teria de existir uma outra dimensão com o número do jogador, de modo a evitar mais tarde código em quadriplicado para os 4 jogadores.

Exemplo 2 - sem erro de qualidade

```
int temAs=0, temAs2=0;
```

Poderia-se aplicar o EQ3 aqui mas o verdadeiro erro é separar cada Ás, e assim teriam de ser considerados 16 Ases que é o que existe em 4 baralhos. A utilização de um vetor de 16 posições, resolvia o EQ3. Mas a agregação dos Áses poderia ser feita aqui, e a maior parte fez, já que interessa apenas saber o número de Ases que estão contabilizados como 11, e podem passar a 1. Como o limite do jogo é 21, por observação de que este valor nunca pode ser 2 (já que somaria 22), poderiam simplificar mas assim se em vez de 21 o limite de pontos for outro, essa parte do código deixa de funcionar, pelo que ficam com código mais específico. Seja como for, há que ter em atenção para oportunidades de agregação de informação, neste caso basta um contador.

Exemplo 3 - EQ9

```
pts[0]='+';

pts[1]='1';

pts[2]='\0';
```

Falha de utilização das funções de strings, aqui seria: strcpy(pts,"+1");

Exemplo 4 - sem erro de qualidade

```
 int mao[100];  // cartas para cada jogador de 20 em 20 posições
```

Juntar num só vetor dados de várias entidades, não faz muito sentido, já que torna a gestão do vetor, neste caso de 20 em 20, do lado do programador, prejudicando a leitura e reutilização do código. Caso extremo desta situação, é alocar um só vetor para todas as variáveis inteiras, e assim estariam a gerir a memória, quando essa é uma tarefa do sistema operativo. Esta é a situação inversa do EQ3, que não inviabiliza a utilização de ciclos, apenas complica expressões que seriam de outra forma mais simples.

Exemplo 5 - Estrutura de dados aconselhada

```
 int limites[5][2] = { {17, 17},  {14, 17},   {16, 18},   {12, 15},   {17, 19
} };
```

Esta estrutura de dados fez toda a diferença, poucos estudantes a utilizaram. As constantes devem naturalmente ser colocadas num vetor de modo a poder utilizar o mesmo código para todos os jogadores. Caso assim não seja, irá existir sempre algum código em quadruplicado. A situação menos grave é quem quadruplica apenas 4 chamadas a uma função, com parâmetros distintos, estendendo-se a situações nem por isso pouco frequentes, em que o código é todo duplicado para cada jogador, não existindo chamadas a funções.

Exemplo 6 - Abstracção aconselhada

```
SolicitarCartas( argumentos conforme a estrutura de dados )

// cada jogador vai pedido mais cartas (chamada a DarCartas),

// até que atinja o seu limite, dependente de ter ou não um Ás a valer 11.
```

Esta é a segunda fase do jogo, se existir alteração de estratégia de jogo, é aqui que se alterava, se os limites tivessem aqui colocados localmente, melhor, poupavam nos argumentos da função.

Exemplo 7 - Abstracção aconselhada

```
ContabilizarResultados(...) // analisa o resultados do jogo, após todos os jo
gadores jogarem
```

Uma função deste tipo ficara também bem, já que as regras de quem ganha/perde poderiam mudar e assim fica encalapsado nesta função. Mas claro que colocando na função Jogar está também tudo bem.

Exemplo 8 - Abstracção aconselhada

```
Jogo( … ) // simula um jogo
```

Como é solicitado vários jogos, é natural que exista uma função deste tipo, simulando um jogo.

Cumprimentos,
José Coelho