# QXCode - Quixadá Coding Team

**Fundamentos de Programação**

**2 de setembro de 2016**

# Blackjack
# Laboratório Prático

David Sena Oliveira*

## 1 Introdução

Nessa prática nós vamos trabalhar apenas com:

- Estruturas de seleção,
- Vetores
- Laço,
- Funções.

Vamos trabalhar com C++ nesse laboratório.

## 2 Primeiras definições

A primeira decisão é como vamos definir uma carta. Podemos utilizar:

- Uma variável **int** de 1 a 13 representando as cartas.
- Uma variável **char** com A1234567890JQK sendo nossas cartas.

Basicamente precisamos ser capazes de sortear, imprimir e manipular nossas cartas. As duas primeiras opções são boas. Vamos escolher definir a carta como um inteiro. [1]

*sena.ufc@gmail.com

[1] Sugiro que você tente fazer com **char** por diversão depois.

## 2.1 Sorteando a carta

Nosso primeiro objetivo é fazer uma função que gere aleatóriamente uma carta válida. Por enquanto, não vamos montar o baralho todo. Vamos apenas gerar um valor válido aleatório entre 1 e 13.

Em C e C++ você pode usar a função `rand()` para gerar um número aleatório.

rand() em C++

```
#include <iostream>
#include <cstdlib> //para usar rand() e srand()
#include <ctime> //para utilizar time()

int main(){
  srand(time(NULL)); //inicia a aleatoriedade
int num = rand() % 5; // 0, 1, 2, 3 ou 4
std::cout << num << std::endl;
return 0;
}
```

Você só precisa chamar a função `srand(time(NULL))` uma única vez no seu código. Normalmente, durante a inicialização das variáveis. Faça uma função `int sortear_carta();`que gera um número entre 1 e 13 usando `rand`. Teste pra ver se ela funciona mesmo e se a cada vez que seu código é executado, ela gera um valor diferente.

# 3 Mostrando a carta

Precisamos ser capazes de mostrar o número como uma carta. Se nossa carta é um 12, queremos imprimir $Q$ e não o 12. Podemos fazer isso de duas

formas. A primeira é criar uma função que recebe o número inteiro e retorna uma `string`. A segunda é colocando todas as cartas em um vetor de strings e utilizar o índice para pegar as cartas.

Mostrando as cartas

```
#include <iostream>
#include <vector>

//Primeira forma
string pegar_nome(int carta){
  if(carta == 1)
    return "A";
  if(carta > 1 && carta < 11)
    return std::to_string(carta);
  //...
}

//Segunda forma:
vector<string> nomes{"", "A", "2", ..., "Q", "K"};
string pegar_nome(int carta){
  return nomes[carta];
}
```

O método `string to_string(value)` to C++ recebe um int, char, float, double e retorna uma string.

## 3.1 Calculando mão de cartas

Vai ser útil se você tiver uma função que recebe a carta e retorna o valor.

```
int pegar_valor(int carta){
  if(carta == 1)
    return 11;
  else if( //termine o codigo ok...
}
```

Para nós, o A vale 11, a não ser que isso estoure a mão do jogador. Mas por favor, não faça um `if` para cada carta, ok? Observe a lógica e veja que

10, J, Q e K, possuem o mesmo valor, portanto podem ficar no mesmo `if`. Os números de 2 a 9 também podem ficar no mesmo `if`.

## 3.2 Calculando a mão

Agora o caldo engrossa! Aqui temos provavelmente a parte mais interessante do código. Nosso objetivo é assumir que o A vale 11 e calcular o valor da mão. Ao mesmo tempo contamos quantos As tem na nossa mão. Após termos o total, se ele passar de 21, vamos descontando nossas cartas As, para fazer o valor baixar.

Pseudocódigo para calcular o valor de uma mao

```
int valor_mao(vector<int> mao){
  total = 0
  n_as = 0
  Para cada carta na mao:
    incremente total do valor da carta
    Se a carta for um A:
      n_as aumenta de 1
  Enquanto o total > 21 e n_as > 0:
    decrementa total de 10
    decremente n_as de 1
  retorne total
}
```

Perceba que fazer o A voltar de 11 para 1 significa retirar 10 pontos do valor da mão. Você pode fazer isso, quantas vezes você precisar(enquanto tiver estourado) e ainda tiver A pra trocar.

## 3.3 Vamos JOGAAAAAAAAARRRRRR!!!

O dealer[2] pede uma carta para mesa e duas para o jogador deixando todas viradas para cima. O jogador vai pedindo cartas à mesa. Seu objetivo é chegar o mais perto que puder de ter uma mão com valor 21. Se ele passar de 21 ele automaticamente perde. Se ele fizer exatos 21 pontos ele ganha.

[2]O funcionário que coordena a mesa

Após o jogador parar sua jogada, supondo que ele não estourou o limite, a mesa(máquina) vai puxar as cartas para tentar vencer o jogador. A mesa tem a vantagem do empate. Se ela fizer a mesma pontuação do jogador ela ganha. Se ela estourar, ela perde. [3]

Vou lhe ajudar com a lógica ok?

Pseudocódigo para lógica do jogo

```
mao_mesa comeca vazia
mao_jogador comeca vazia
puxa uma carta para mao_mesa
puxa duas cartas para mao_jogador
PARAR = false
Enquanto jogador nao PARAR ou nao estourar:
  mostrar maos
  pergunta opcao ao jogador
  se opcao for pedir
    puxa carta para mao_jogador
  senao se opcao for parar
    PARAR = true
Enquanto mesa nao ganhar ou estourar
  puxar carta para mesa
  mostrar maos
```

Sugiro que crie uma função que mostre a mão de um jogador. Algo como `string mostrar_mao(vector<int> mao)`. A saída pode ser parecida `Total 17 [2 A 4 K]`. A seguir, um exemplo de possível aparência do jogo.

[3] No nosso modelo, a mesa ou ganha ou estoura, porque ela não vai se contentar em perder.

Exemplo: Jogador perdendo.

```
Iniciando Rodada:
# Mesa recebe  7 - Total  7 [ 7 ]
# Voce recebe  A - Total 11 [ A ]
# Voce recebe  2 - Total 13 [ A 2 ]
Pedir = 1, Parar = 2
>> 1

# Voce recebe  3 - Total 16 [ A 2 3 ]
Pedir = 1, Parar = 2
>> 2

# Mesa recebe  2 - Total  9 [ 7 2 ]
# Mesa recebe  7 - Total 16 [ 7 2 7 ]
# Mesa (16), Voce (16)
Voce perdeu!
```

# 4 Melhorias

Deixo aqui algumas sugestões pra sua diversão.

1. Implementar um laço no qual o jogador continua jogando até que decida parar.

2. Implementar um esquema de apostas no qual o jogador decide quanto quer apostar antes da rodada. O jogador pode começar com uma quantia fixa de dinheiro. Se ele ganhar a rodada, o dinheiro é adicionado na conta dele. Se ele perde, então, a, um, então ele perde.

3. Montar um ou dois baralhos com as cartas e embaralhar. Retirar então as cartas, e após a rodada, elas vão para o montante até que o baralho termine.

4. Implementar o jogo com mais de um jogador. Nele, a mesa vai precisar de regras um pouco diferentes. Consulte a wiki sobre Blackjack para entender mais.