CURSO SUPERIOR DE ADS

Arranjos e Números pseudoaleatórios

Prof. Fernando Marlon Soares Figueiredo

Disciplina: Programação Estruturada

SECTRAS

ADS

Aula de Hoje

- Arranjos
- Números pseudoaleatórios

Introdução

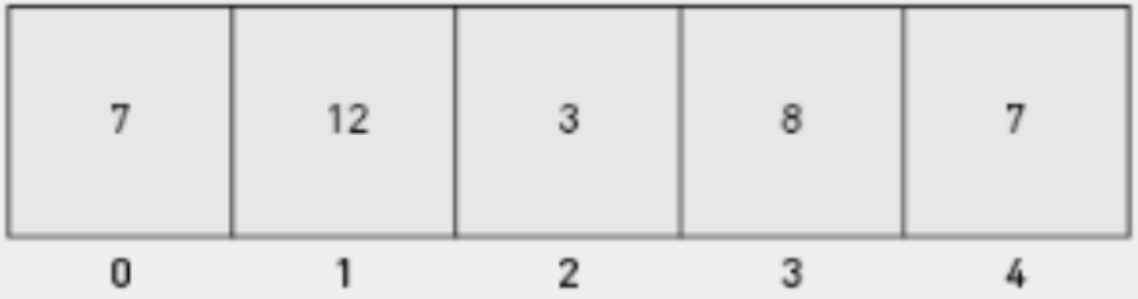
- Até então, com os conceitos vistos, não há uma forma eficiente de armazenar um volume maior de dados.
- Se quiséssemos fazer um programa para armazenar 50 notas de alunos, com o conhecimento visto até aqui, teríamos que criar 50 variáveis de notas e registrar uma a uma.
- Além de sujeito a erros, este processo se tornaria inviável caso o número de notas a ser registrado fosse muito maior.

Arranjos (ou Arrays)

- São estruturas que permitem o agrupamento de dados, sendo referenciados por uma variável.
- Os dados devem ser homogêneos, ou seja, todos os elementos devem ser de um mesmo tipo e sua estrutura é indexada, fazendo com que os elementos possam ser diretamente acessados pelo seu índice.
- Arranjos podem possuir uma ou mais dimensões, considerando a forma em que os dados estão dispostos.

Vetores – Arranjos de uma dimensão

- A forma mais utilizada de arranjos é no formato unidimensional, também chamados de vetores. Em um vetor, dados são organizados e recuperados pelo acesso ao seu índice, que é único.
- Imagine o vetor como uma rua, os índices como as casas dessa rua e os elementos são os moradores das casas.



Sintaxe

- tipo nome [tamanho]

Onde:

- Tipo corresponde ao tipo de dados.
- Nome ao identificador da variável.
- Tamanho corresponde a quantidade de elementos do vetor.

Exemplo:

float notas[5]

Lixo de Memória

- Quando um array é declarado, não é feito qualquer tipo de limpeza nessa região da memória alocada. Portanto, assume-se que os valores dos elementos são desconhecidos, sendo chamados de **lixo de memória**.
- Dessa forma, é indicado que os vetores sejam inicializados (atribuídos com valores desejados) antes de seu acesso.

Atribuindo valores ao vetor

tipo nome[tamanho] = { e1, e2, e3, ..., eN };

- Onde e1, e2, e3... E em são elementos do tipo especificado .
- N corresponde ao tamanho do vetor.

- Também podemos alterar os elementos por meio dos índices:

nome[índice]

Exemplo

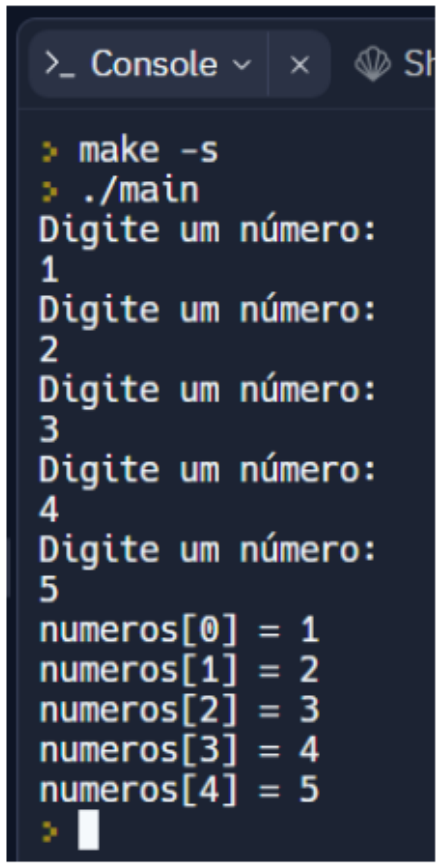
```
1 #include <stdio.h>
2 #define NUM_ALUNOS 5
3
4 int main
5
```

```
6     float notas[NUM_ALUNOS] ;
7     int i;
8
9     for (i =0; i <NUM_ALUNOS; i++)
10     {
11             printf("Digite a nota do aluno %d :", i);
12             scanf("%f", &notas[i]);
13     }
```

```
15     printf("As notas digitadas foram: \n");
16
17     for (i =0; i <NUM_ALUNOS; i++)
18             printf("Nota[%d ] = %f", i , notas[i]);
19
20     return 0;
21 }
```

Exercício 1

- Faça um programa que peça ao usuário para digitar 5 números inteiros e armazene-os em um vetor.
- Ao final, imprima os elementos desse vetor.



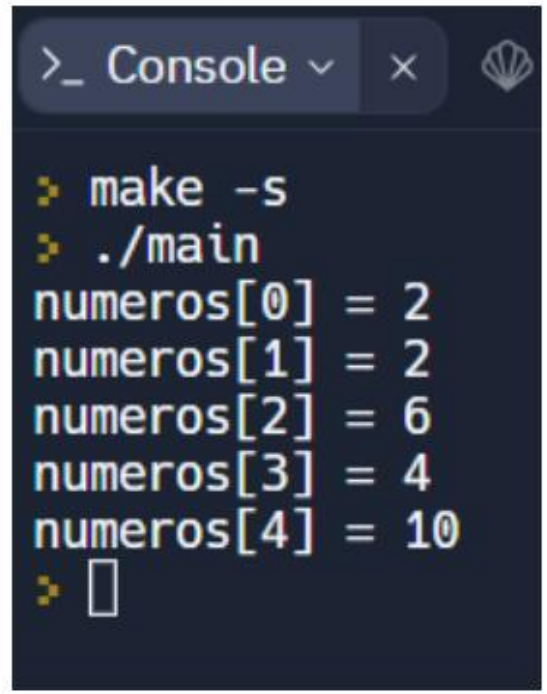
Exercício 1 - Resposta

```c
#include <stdio.h>
#define NUMMAX 5
int main(void) {

    int numeros[5];
    for (int i=0; i<NUMMAX; i++) {
        printf("Digite um número:\n");
        scanf("%d", &numeros[i]);
    }

    for (int i=0; i<NUMMAX; i++) {
        printf("numeros[%d] = %d\n", i, numeros[i]);
    }

    return 0;
}
```

```
>_ Console
> make -s
> ./main
Digite um número:
1
Digite um número:
2
Digite um número:
3
Digite um número:
4
Digite um número:
5
numeros[0] = 1
numeros[1] = 2
numeros[2] = 3
numeros[3] = 4
numeros[4] = 5
>
```

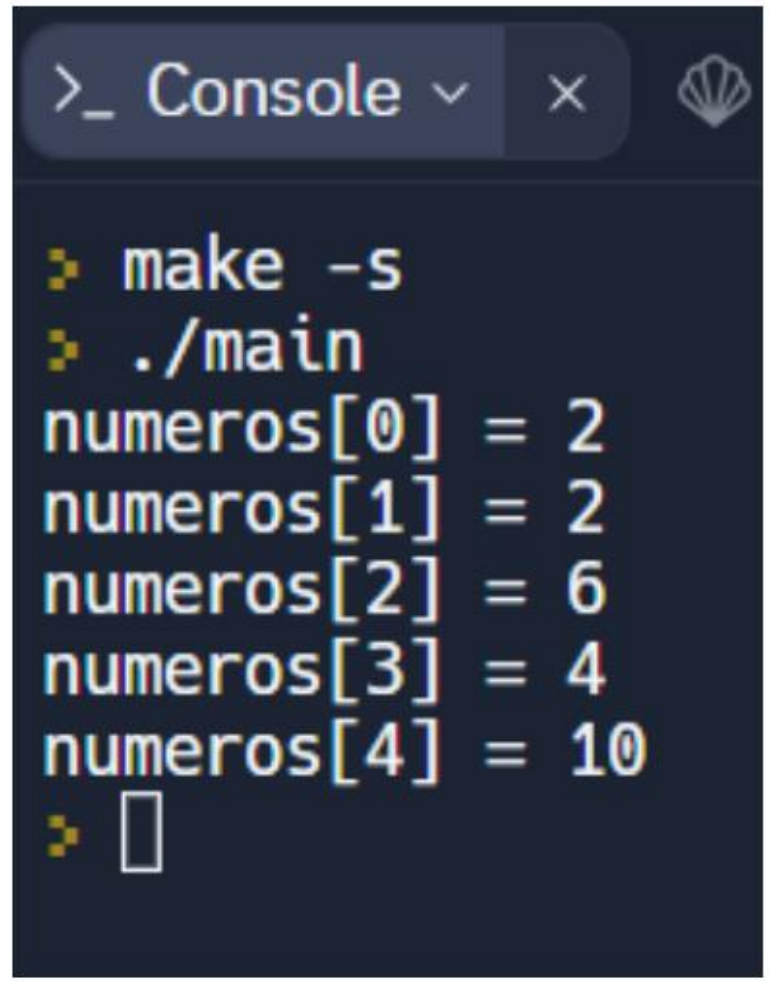
Exercício 2

- Faça um programa que crie um vetor com cinco números inteiros preenchidos pelo programador na declaração.
- Em seguida altere todos os números dos índices pares (incluindo o zero) para que seu valor seja multiplicado por dois.
- Ao final, imprima o vetor resultante.



Exercício 2 - Resposta

```c
#include <stdio.h>
#define NUMMAX 5
int main(void) {
    int numeros[5] = {1, 2, 3, 4, 5};
    for (int i=0; i<NUMMAX; i++) {
        if (i%2==0) {
            numeros[i]*=2;
        }
    }

    for (int i=0; i<NUMMAX; i++) {
        printf("numeros[%d] = %d\n", i, numeros[i]);
    }
    return 0;
}
```

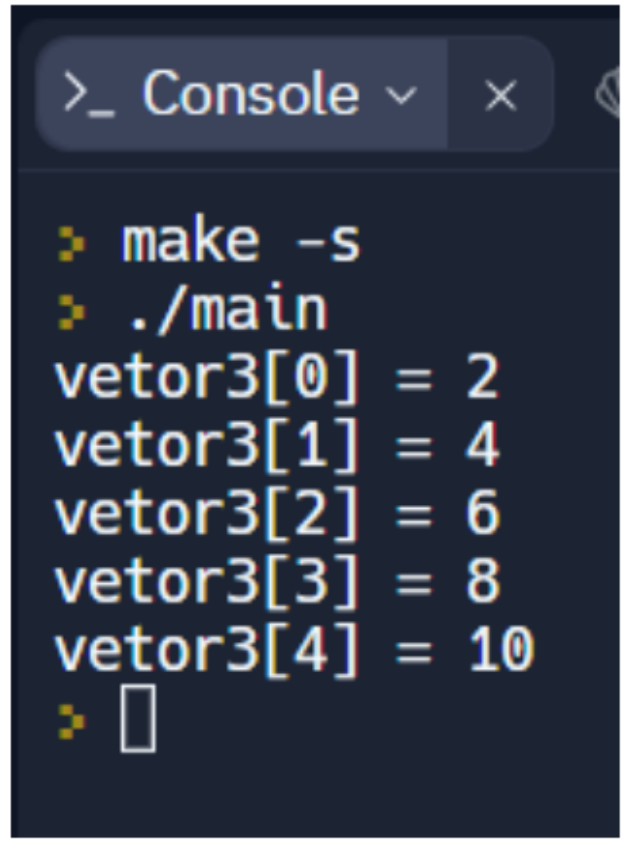


Exercício 3

- Faça um programa que contenha dois vetores de inteiros com cinco elementos cada, sendo preenchidos pelo programador durante a declaração.
- O programa deve gerar um terceiro vetor onde cada posição é formada pela soma dos elementos das respectivas posições dos dois vetores.
- Ao final, imprima os valores do vetor gerado.

Exercício 3 - Resposta

```c
#include <stdio.h>
#define NUMMAX 5
int main(void) {
    int vetor1[5] = {1, 2, 3, 4, 5};
    int vetor2[5] = {1, 2, 3, 4, 5};
    int vetor3[5];
    for (int i=0; i<NUMMAX; i++) {
        vetor3[i] = vetor1[i]+vetor2[i];
    }

    for (int i=0; i<NUMMAX; i++) {
        printf("vetor3[%d] = %d\n", i, vetor3[i]);
    }
    return 0;
}
```



Matrizes – Arranjos de duas dimensões

- Dispõem os dados em formato de tabela, possuindo dois índices que indicam, respectivamente, linha e coluna.
- A sintaxe para declaração e acesso as matrizes se dá de forma análoga à de vetores.

- tipo nome[tamanho linha] [tamanho coluna]; //declaração
- nome[linha][coluna] //acesso

Arranjos de Múltiplas dimensões

- É possível criar arranjos de múltiplas dimensões, bastando ampliar o conceito, ou seja, adicionando mais uma dimensão na declaração.

- tipo nome[tamanho 1][tamanho 2]... [tamanho N]; //declaração
- Nome[índice1][índice 2]... [índice N]; //acesso

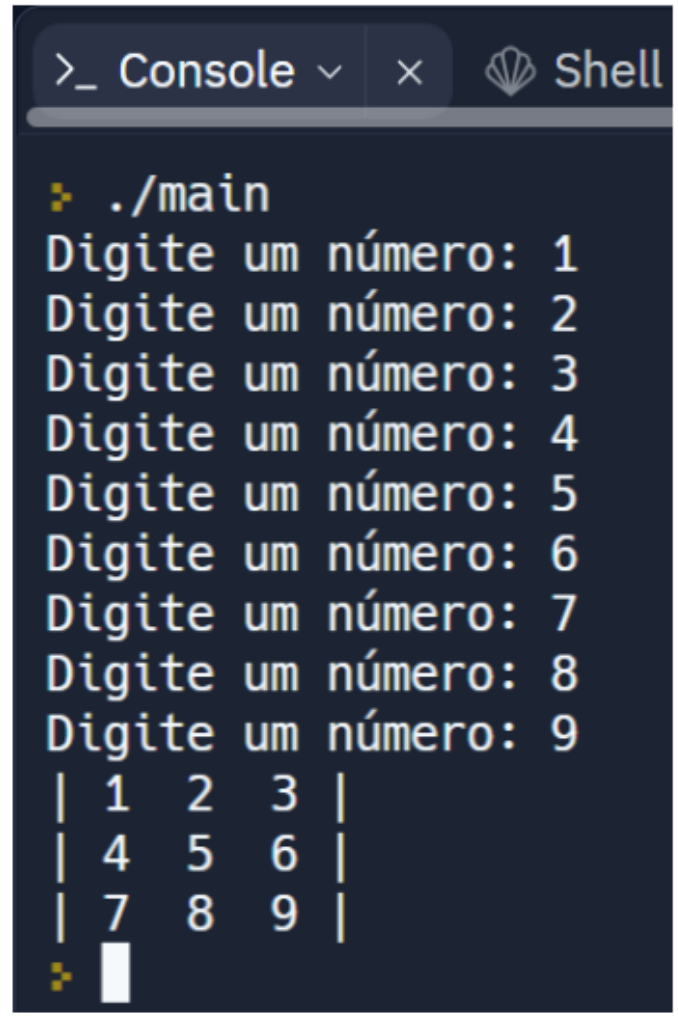
- Geralmente, costuma-se evitar arranjos de múltiplas dimensões, uma vez que tornam o código mais confuso.

Pontos Importantes sobre Arranjos

1. Os índices sempre iniciam com zero e terminam em tamanho -1.
2. Acesso a elementos de índices inexistentes (fora dos limites do vetor) em arranjos causam erros de execução, interrompendo o programa abruptamente. Esse ponto é de inteira responsabilidade do programador.
3. O tamanho dos arranjos deve ser preenchido obrigatoriamente no momento da declaração, ou seja, ainda durante a fase de compilação do código.

Exercício 4

- Faça um programa que peça para o usuário preencher os elementos de uma matriz 3x3 de inteiros e imprima na tela a **matriz formatada** e o maior elemento.



Exercício 4 - Resposta

```c
#include <stdio.h>
#define NUMCOL 3
#define NUMLIN 3
int main(void) {
  int matriz[3][3];
  int maior = 0;
  for (int i=0; i<NUMLIN; i++) {
    for (int j=0; j<NUMCOL; j++) {
      printf("Digite um número: ");
      scanf("%d", &matriz[i][j]);
      if (matriz[i][j] > maior) {
        maior = matriz[i][j];
      }
    }
  }
  for (int i=0; i<NUMLIN; i++) {
    printf("|");
    for (int j=0; j<NUMCOL; j++) {
      printf(" %d ",matriz[i][j]);
    }
    printf("|\n");
  }
  printf("O maior número é %d", maior);

  return 0;
}
```

Números Pseudoaleatórios

Como são gerados?

- Números aleatórios são aqueles escolhidos ao acaso, de acordo com uma certa função de probabilidade. Apesar de serem chamados aleatórios, qualquer número computacional na verdade não atende aos conceitos de aleatoriedade matemáticos, onde a chance de serem sorteados deve ser completamente independente de fatores externos.
- Portanto, em computação, temos números chamados pseudoaleatórios, que serão gerados por uma função matemática pré-determinada, geralmente calculada em função do tempo atual do relógio do computador.

Precisamos de 3 funções para criar números pseudoaleatórios

time()

- Retorna o tempo total em segundos desde a meia-noite do dia primeiro de janeiro de 1970.
- Faz parte da biblioteca time.h.

Precisamos de 3 funções para criar números pseudoaleatórios

srand()

- Responsável por inicializar a semente (seed) do gerador de números aleatórios. Utilizado em conjunto com a função **time** para garantir que a cada execução seja gerado um valor diferente.
- Faz parte da biblioteca stdlib.h.

Precisamos de 3 funções para criar números pseudoaleatórios

rand()

- Responsável por gerar um número inteiro aleatório entre 0 e uma constante própria da linguagem chamada de RAND MAX. Para garantir que o número gerado seja calculado a partir de uma nova semente a cada execução, é necessário ter inicializada a semente com a função srand.
- Faz parte da biblioteca stdlib.h.

Aplicando as 3 funções...

- É possível gerar um número inteiro que provavelmente será diferente a cada execução, mas ainda fica restrito aos valores 0 e RAND MAX.
- Uma forma de adaptar esse valor para um intervalo mínimo e máximo é utilizando o resto da divisão inteira com o operador % por meio da seguinte fórmula.

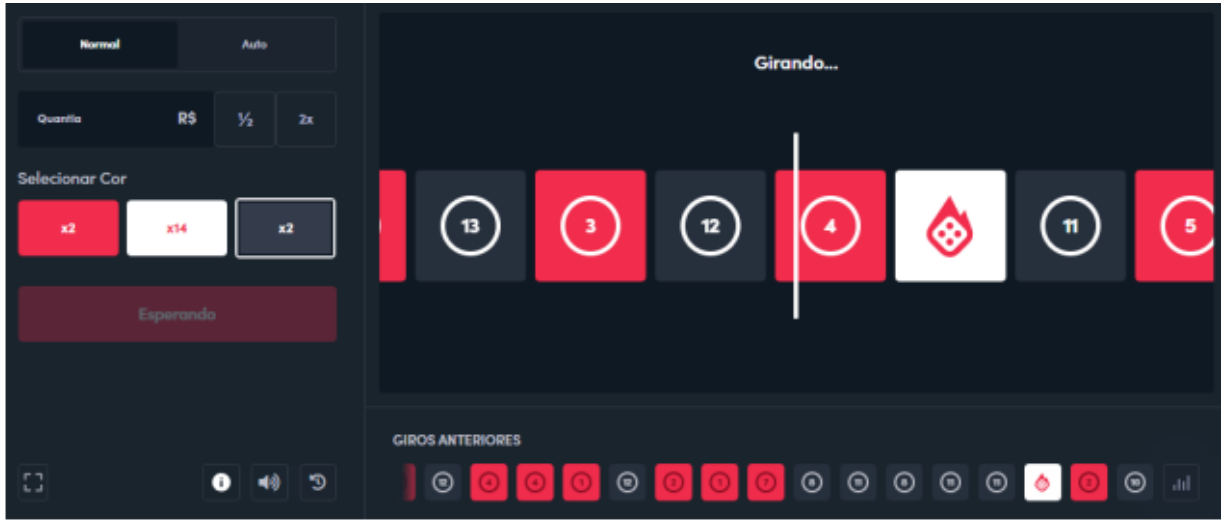
```
MIN + (rand( ) % (MAX – MIN + 1));
```

- Onde MIN e MAX são os valores desejados para o intervalo.

Exemplo

```c
#include <stdio.h>
#include <stdlib.h> // para funcoes rand ( ) e srand( )
#include <time.h> // para funcao time ( )
#define MIN 1
#define MAX 20

int main( )
{
    int d20;

    srand(time(NULL));

    printf("Rolando D20 ... \ n");
    d20 = MIN + (rand ( ) % (MAX − MIN + 1));
    printf("Resultado: %d\n", d20);

    return 0;
}
```

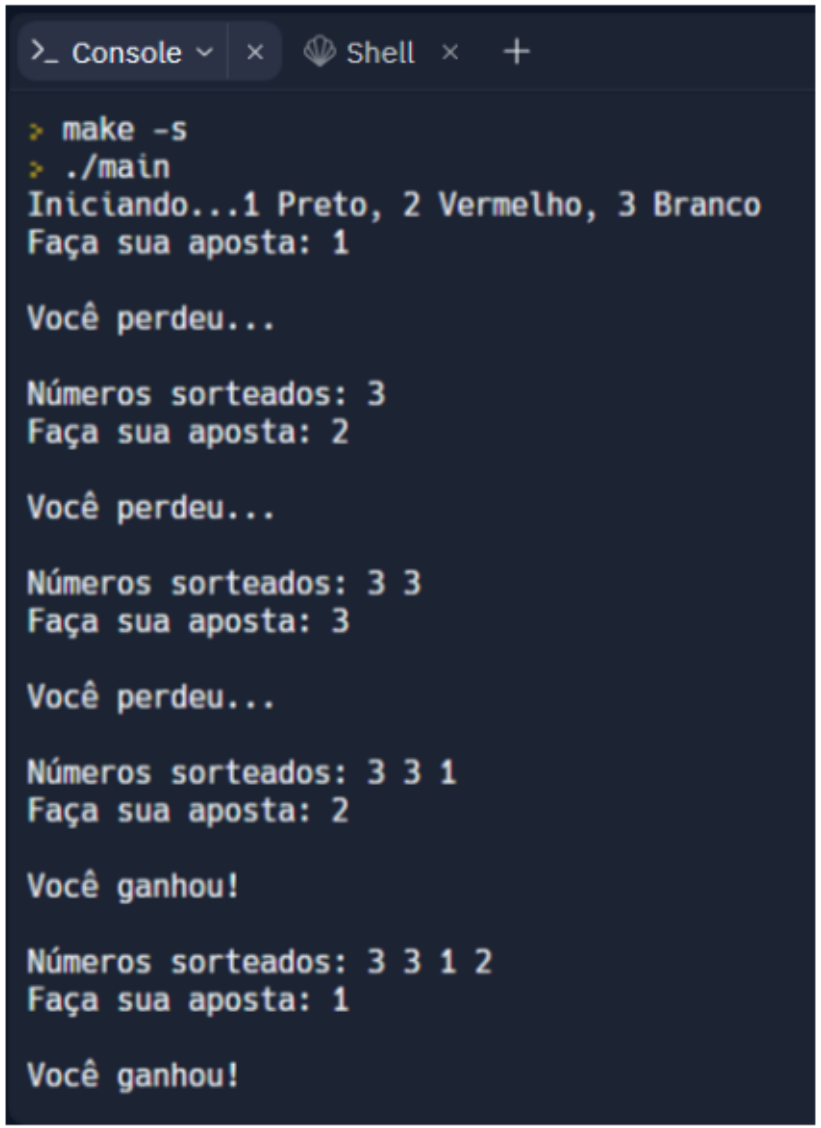
Exercício 5



- Faça um programa para sortear números de 1 a 3. Simulando as cores do Blaze (vermelho, preto ou branco).
- Pergunte ao usuário qual a escolha dele (entre 1 e 3). Na sequencia faça o sorteio e diga se o usuário ganhou ou perdeu.

- Devem ser realizados 10 sorteios. A cada sorteio, o número deve ser salvo num vetor de resultados.

Exemplo do console



Exercício 5 - Resposta

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MIN 1
#define MAX 3
int main(void) {
  int sorteados[10] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1};
  int sorteio, aposta;
  srand(time(NULL));
  printf("Iniciando...");
  printf("1 Preto, 2 Vermelho, 3 Branco");
  for (int i=0; i<10; i++) {
    sorteio = MIN + (rand()%(MAX - MIN + 1));
    sorteados[i] = sorteio;
    printf("\nFaça sua aposta: ");
    scanf("%d", &aposta);
    if (aposta==sorteio) {
      printf("\nVocê ganhou!");
    } else {
      printf("\nVocê perdeu...");
    }
    printf("\n\nNúmeros sorteados: ");
    for (int j=0; j<i+1; j++) {
      printf("%d ", sorteados[j]);
    }
  }
  return 0;
```

Exercício 6

- Faça um programa que pede para o usuário preencher um valor de seis posições de inteiros com valores entre 1 e 60.
- Após preenchidos, gere seis números aleatórios e indique quantos o usuário acertou.
- Não há necessidade de conferir que os números sejam únicos.