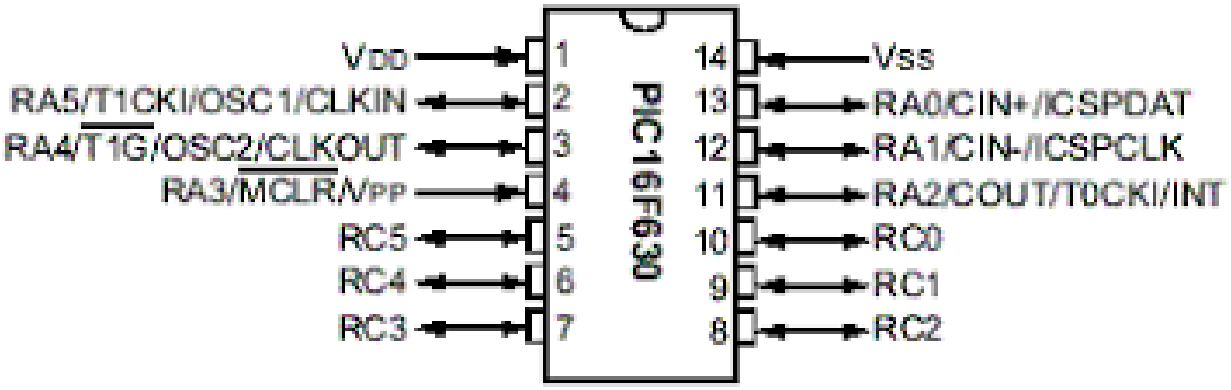
PLANO DE AULA

PIC16F630:

O PIC16F630, fabricado pela Microchip, é um microcontrolador de 8 bits amplamente utilizado devido à sua simplicidade, custo acessível e versatilidade. Ele é ideal para projetos que envolvem entrada e saída de dados digitais, lógica de programação básica e até mesmo funcionalidades mais avançadas, como temporização e interrupções. A seguir, destacamos os aspectos mais relevantes do microcontrolador, incluindo suas características de pinagem.



Assim como outros modelos da série PIC, o PIC16F630 possui uma disposição de pinos que oferece funcionalidades diversas. Abaixo, explicamos os principais pinos e suas utilidades com base no que será comumente utilizado:

- **VDD (Alimentação):** Este pino é responsável por receber a fonte de alimentação do microcontrolador, que normalmente opera com uma tensão de 5V. Ele fornece a energia necessária para o funcionamento interno do dispositivo.
- **VSS (GND – Ground):** O pino de referência de terra (ground), essencial para completar o circuito elétrico e fornecer um referencial de tensão estável ao microcontrolador.
- **RA (PORTA) e RC (PORTC) – Pinos de Entrada e Saída de Dados:**

    - Esses conjuntos de pinos (PORTA e PORTC) são dedicados à comunicação de dados e podem ser configurados como entrada ou saída digital, dependendo da aplicação.
    - Cada pino possui funções específicas que podem ser habilitadas ou configuradas por meio de registradores internos.
    - Além das funções digitais, alguns desses pinos podem desempenhar papéis adicionais, como entradas analógicas, saídas de PWM (modulação por largura de pulso) ou sinais de clock.

Características Básicas do PIC16F630

O PIC16F630 é um microcontrolador compacto, mas potente, com as seguintes características principais:

- **Clock Interno:** Possui um oscilador interno configurável, eliminando a necessidade de componentes externos para geração de clock em muitos projetos.
- **Memória:** Integra memória flash para armazenamento do programa, memória RAM para dados voláteis e EEPROM para dados permanentes.
- **I/O Versátil:** Com 12 pinos disponíveis para entrada e saída de dados, o microcontrolador oferece flexibilidade para diversos projetos.
- **Periféricos:** Inclui temporizadores, comparadores analógicos e outros recursos que ampliam suas possibilidades de uso.
- **Baixo Consumo de Energia:** Pode operar com eficiência em sistemas alimentados por baterias.

  Link datasheet:
  https://ww1.microchip.com/downloads/en/devicedoc/40039c.pdf

Exercícios:

Projeto: Divisão com Verificação de Divisor

Nesta atividade prática, utilizaremos conceitos fundamentais de programação em C, como entrada e saída de dados, operações matemáticas e estruturas condicionais. O objetivo do exercício é criar um programa simples que leia dois números inteiros, calcule o quociente e o resto da divisão entre eles e exiba os resultados. Além disso, adicionaremos uma verificação para garantir que o divisor nunca seja zero, evitando erros de execução.

Descrição do Programa

1. **Leitura de Dados:**
    - O programa solicita ao usuário dois números inteiros: um numerador e um divisor.
    - Garantimos que o divisor será diferente de zero para que a operação de divisão seja válida.
2. **Cálculo da Divisão:**
    - Com os números fornecidos, o programa calcula:
        - O **quociente** (resultado da divisão inteira).
        - O **resto** da divisão.
3. **Exibição dos Resultados:**
    - Caso o divisor seja válido, os resultados do quociente e do resto são exibidos.
    - Caso o divisor seja zero, o programa exibe uma mensagem de erro, informando que a operação não é permitida.
4. **Estrutura Condicional:**
    - A lógica do programa utiliza um bloco `if` para verificar se o divisor é diferente de zero antes de realizar os cálculos. Isso ilustra como a programação pode lidar com condições específicas para evitar erros.

```
#include <stdio.h>

int main() {
	int num1, num2, quociente, resto;

	printf("Digite o primeiro numero inteiro: ");
	scanf("%d", &num1);

	printf("Digite o segundo numero inteiro (diferente de zero): ");
	scanf("%d", &num2);

	if (num2 != 0) {
	quociente = num1 / num2;
	resto = num1 % num2;
```

```
        printf("Quociente: %d\n", quociente);
        printf("Resto: %d\n", resto);
        } else {
        printf("Erro: O divisor nao pode ser zero.\n");
        }

        return 0;
}
```

Projeto: Calculadora de Quociente e Resto

Neste exercício, vamos explorar o uso de entrada e saída de dados, estruturas condicionais e operações matemáticas fundamentais na linguagem C. O objetivo do programa é permitir que o usuário insira dois números inteiros e, em seguida, exibir o **quociente** e o **resto** da divisão do primeiro pelo segundo. Para evitar problemas, o programa valida se o divisor é diferente de zero antes de realizar os cálculos.

Objetivo do Exercício

Este programa tem como metas:

1. Introduzir conceitos de divisão inteira e cálculo de restos.
2. Demonstrar a importância da validação de entradas em um programa.
3. Explorar o uso de condições para prevenir erros de execução.

Descrição do Programa

1. **Entrada de Dados:**
    - O programa solicita dois números inteiros:
        - O primeiro número será o **dividendo**.
        - O segundo número será o **divisor** (deve ser diferente de zero).
2. **Validação:**
    - Antes de realizar os cálculos, o programa verifica se o divisor é zero.
    - Caso o divisor seja inválido (zero), o programa exibe uma mensagem de erro e encerra a execução.
3. **Cálculo e Saída:**
    - Se o divisor for válido, o programa calcula:
        - O **quociente** da divisão inteira entre os dois números.
        - O **resto** da divisão.
    - Os resultados são exibidos na tela.

```
#include <stdio.h>
```

```
int main() {
	int num1, num2, quociente, resto;

	printf("Digite o primeiro numero inteiro: ");
	scanf("%d", &num1);

	printf("Digite o segundo numero inteiro (diferente de zero): ");
	scanf("%d", &num2);

	if (num2 != 0) {
	quociente = num1 / num2;
	resto = num1 % num2;

	printf("Quociente: %d\n", quociente);
	printf("Resto: %d\n", resto);
	} else {
	printf("Erro: O divisor nao pode ser zero.\n");
	}

	return 0;
}
```

Projeto: Área do círculo

Neste exercício, os alunos irão aprender a calcular a **área de um círculo** com base no valor do raio fornecido pelo usuário. O programa utiliza uma constante para o valor de **π (PI)**, valida a entrada do usuário e aplica a fórmula matemática da área do círculo.

Objetivo do Exercício

- Demonstrar o uso de constantes em C.
- Utilizar a função `pow` da biblioteca `math.h` para realizar cálculos exponenciais.
- Ensinar como validar entradas de dados e exibir mensagens claras para o usuário.
- Aplicar operações matemáticas em um programa prático.

Descrição do Programa

1. **Entrada de Dados:**
    - O programa solicita ao usuário o valor do **raio** do círculo.
2. **Validação:**

    - Verifica se o valor do raio é positivo.
    - Caso contrário, informa que o raio deve ser um número positivo.
3. **Cálculo e Saída:**
    - Se o raio for válido, o programa calcula a **área** do círculo usando a fórmula: A´rea=π×(raio)2\text{Área} = \pi \times (\text{raio})^2A´rea=π×(raio)2
    - Exibe a área do círculo com duas casas decimais.

```
#include <stdio.h>
#include <math.h>

int main() {
	float raio, area;
	const float PI = 3.14159; // Definição da constante PI

	printf("Digite o valor do raio do circulo: ");
	scanf("%f", &raio);

	if (raio > 0) {
	area = PI * pow(raio, 2); // Cálculo da área
	printf("A area do circulo com raio %.2f eh: %.2f\n", raio, area);
	} else {
	printf("O raio deve ser um valor positivo. Por favor, tente novamente.\n");
	}

	return 0;
}
```

Projeto: Calculadora

Neste exercício, os alunos desenvolverão uma **calculadora interativa** que realiza as quatro operações básicas: multiplicação, divisão, subtração e soma. O programa também valida as entradas e fornece mensagens informativas em caso de erros, como divisão por zero ou seleção de opção inválida.

Objetivo do Exercício

- Ensinar a utilização da estrutura `switch-case` para controle de fluxo.
- Demonstrar como validar entradas do usuário.
- Aplicar operações matemáticas básicas em um programa interativo.
- Incentivar boas práticas de programação, como mensagens claras e controle de erros.

Descrição do Programa

1. **Menu de Opções:**
    - O programa apresenta um menu com as opções:
        - 1: Multiplicação.
        - 2: Divisão.
        - 3: Subtração.
        - 4: Soma.
2. **Entrada de Dados:**
    - O usuário escolhe uma opção do menu e fornece dois números reais para realizar a operação.
3. **Validação e Operações:**
    - Dependendo da opção selecionada, o programa realiza a operação correspondente:
        - Verifica se o divisor é diferente de zero no caso da divisão.
        - Exibe mensagem de erro se o usuário inserir uma opção inválida.
4. **Saída:**
    - O resultado da operação é exibido com duas casas decimais.

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

int main() {
	float numero1, numero2, resultado;
	int escolha;

	printf("\t-- Calculadora! --\n");
	printf("\tMultiplicacao - 1\n");
	printf("\tDivisao ------- 2\n");
	printf("\tSubtracao ----- 3\n");
	printf("\tSoma ---------- 4\n");
	printf("\t-- Escolha uma opcao! --\n");
	scanf("%d", &escolha);

	printf("Digite o valor do primeiro numero: \n");
	scanf("%f", &numero1);
	printf("Digite o valor do segundo numero: \n");
	scanf("%f", &numero2);

	switch (escolha) {
	case 1:
	resultado = numero1 * numero2;
```

```
        printf("Resultado: %.2f\n", resultado);
        break;

        case 2:
        if (numero2 != 0) {
                resultado = numero1 / numero2;
                printf("Resultado: %.2f\n", resultado);
        } else {
                printf("Erro: Divisao por zero nao permitida.\n");
        }
        break;

        case 3:
        resultado = numero1 - numero2;
        printf("Resultado: %.2f\n", resultado);
        break;

        case 4:
        resultado = numero1 + numero2;
        printf("Resultado: %.2f\n", resultado);
        break;

        default:
        printf("Opcao invalida. Por favor, escolha entre 1 e 4.\n");
        break;
        }

        return 0;
}
```

Projeto: Altura Minima

Carlitos é um entusiasta de aventuras com um amor insaciável por parques de diversões. Apesar da sua paixão vibrante, Carlitos enfrenta um desafio ´unico: a sua estatura limitada. Enquanto planeja ansiosamente sua aventura de fim de semana, ele percebe que suas limita¸c˜oes verticais podem atrapalhar sua experiˆencia no parque de diversões. Não se trata apenas de escolher um parque; trata-se de encontrar um onde ele possa aproveitar a emoção dos brinquedos. Imagine o caleidoscópio de cores, as risadas jubilosas e a adrenalina dos passeios. Carlitos sempre foi atraído pela energia dos parques de diversões. Com o fim de semana se aproximando, ele se debruça sobre os folhetos do parque, estudando os requisitos de altura de cada passeio. O objetivo dele ´e maximizar sua diversão, e é aí que vocˆe entra. Sua tarefa é ajudar Carlitos a determinar o número de passeios que ele pode desfrutar em um parque específico. Considerando sua altura e os requisitos

mínimos de altura de cada passeio, orientado a aproveitar ao máximo sua aventura no parque de diversões. Entrada A primeira linha contém dois números inteiros, N e H (1 ≤ N ≤ 6 e 90 ≤ H ≤ 200), que representam a quantidade de brinquedos em um parque e a altura de Carlitos em centímetros, respectivamente. Na segunda linha da entrada, serão fornecidas as alturas mínimas A1, . . . , AN (90 ≤ Ai ≤ 200) de cada um dos brinquedos do parque.
Saida programa deve imprimir uma única linha contendo a quantidade de brinquedos nos quais Carlitos pode ir, ou seja, a quantidade de brinquedos para os quais a altura de Carlitos é pelo menos tão grande quanto a altura mínima necessária.

```
#include <stdio.h>
#define MAX 6

int main() {
    int N, H, i, count = 0;
    int alturas[MAX];

    // Lê o número de brinquedos e a altura de Carlitos
    scanf("%d %d", &N, &H);

    // Lê as alturas mínimas dos brinquedos
    for (i = 0; i < N; i++) {
        scanf("%d", &alturas[i]);
        if (H >= alturas[i]) {
            count++;
        }
    }

    // Imprime o resultado
    printf("%d\n", count);

    return 0;
}
```

If e else -

Escreva um programa que pergunte o dia, mês e ano do aniversário de uma pessoa e diga se a data é válida ou não. Caso não seja, diga o motivo. Suponha que todos os meses tem 31 dias e que estejamos no ano de 2024.

```
#include <stdio.h>

int main()
{
   int dia,
      mes,
      ano;

   printf("Dia: ");
   scanf("%d", &dia);

   printf("Mes: ");
   scanf("%d", &mes);

   printf("Ano: ");
   scanf("%d", &ano);


   if((dia < 1) || (dia > 31))
      printf("Dia invalido\n");
   else //se o dia for válido
      if( (mes < 1) || (mes > 12) )
         printf("Mes invalido\n");
      else // além do dia, o mês for válido
         if( ano > 2024 )
            printf("Ano invalido\n");
         else //se além do dia e mês, o ano for válido
            printf("Data valida\n");


}
```

Problema BIT Else

Escreva um programa em C que, dado um número inteiro, conte quantos bits estão setados (com valor 1) em sua representação binária. Mas que não permita dois “1” em sequência no binário.

```
#include <stdio.h>

int count_set_bits(int num) {
    int count = 0;
    while (num) {
        count += num & 1;
        num >>= 1;
    }
    return count;
}

int main() {
    int number;

    printf("Digite um número inteiro: ");
    scanf("%d", &number);

    int set_bits = count_set_bits(number);
    printf("O número de bits setados é: %d\n", set_bits);

    return 0;
}
```

Problema de ponteiros e endereços de memória

Implemente um programa em C que utilize um ponteiro para manipular uma variável inteira. O programa deve:

1. Declarar uma variável inteira e inicializá-la com um valor fixo.
2. Declarar um ponteiro para armazenar o endereço dessa variável.
3. Exibir o valor e o endereço da variável usando tanto a variável quanto o ponteiro.
4. Modificar o valor da variável utilizando o ponteiro e exibir o novo valor.

Requisitos:

- Use o operador de endereço (&) para armazenar o endereço da variável no ponteiro.
- Use o operador de desreferência (*) para acessar e modificar o valor da variável através do ponteiro.

Saída Esperada:
O programa deve exibir o valor inicial da variável, seu endereço, o valor armazenado no ponteiro, o valor apontado pelo ponteiro e o novo valor da variável após a modificação feita através do ponteiro.

Código:

```
#include <stdio.h>

int main() {
    // Declaração de variáveis
    int num = 10;
    int *ptr;  // Declaração de um ponteiro

    // Atribuindo o endereço de num ao ponteiro
    ptr = &num;

    // Exibindo valores e endereços
    printf("Valor de num: %d\n", num);
    printf("Endereço de num: %p\n", &num);

    printf("Valor armazenado em ptr (endereço de num): %p\n", ptr);
    printf("Valor apontado por ptr: %d\n", *ptr);

    // Modificando o valor de num através do ponteiro
    *ptr = 20;
    printf("Novo valor de num após modificação via ponteiro: %d\n", num);

    return 0;
}
```

Problema de matrizes

Implemente um programa em C que crie uma matriz 3x3 de números inteiros e a preencha com valores fornecidos pelo usuário. O programa deve:

1. Declarar uma matriz 3x3.
2. Preencher a matriz com valores informados pelo usuário.

3. Exibir a matriz completa na tela.
4. Solicitar ao usuário uma nova posição (linha e coluna) e um novo valor para atualizar na matriz.
5. Atualizar a matriz com o novo valor fornecido.
6. Exibir novamente a matriz atualizada.

Saída esperada:

Digite os valores para a matriz 3x3:

1 2 3

4 5 6

7 8 9

Matriz 3x3:

1 2 3

4 5 6

7 8 9

Digite a posição (linha e coluna) para alterar: 1 1

Digite o novo valor: 99

Matriz atualizada:

1 2 3

4 99 6

7 8 9

Requisitos:

- Utilize loops aninhados para preencher e exibir a matriz.
- Certifique-se de validar a entrada para garantir que a posição fornecida esteja dentro dos limites da matriz (0 a 2 para linhas e colunas).

Código:

```
#include <stdio.h>

int main() {
    // Declaração e inicialização de uma matriz 3x3
    int matriz[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    // Exibindo a matriz
    printf("Matriz 3x3:\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }

    // Alterando um elemento da matriz
    matriz[1][1] = 99;

    // Exibindo a matriz atualizada
    printf("\nMatriz atualizada:\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```