Calculadora de conjuntos numéricos

None

None

None

Table of contents

1. Documentação

1.1 Bibliotecas e definições

1.1.1 Bibliotecas

São importadas bibliotecas padrão para:

- **Entrada e saída de dados** (`stdio.h`)
- **Alocação dinâmica de memória** (`stdlib.h`)
- **Uso de tipos booleanos** (`stdbool.h`)
- **Funções específicas do Windows**, como a definição do código de página UTF-8 no console (`windows.h`).

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <windows.h>
```

1.1.2 Macros max e min

Definem funções inline para calcular o valor máximo e mínimo entre dois valores, permitindo a reutilização dessas operações em diversas partes do código de forma otimizada.

```c
#define max(a, b) (((a) > (b)) ? (a) : (b))
#define min(a, b) (((a) < (b)) ? (a) : (b))
```

1.1.3 Códigos de Cores

Define códigos de cor ANSI para estilizar a saída no console. As cores disponíveis são:

- **Vermelho** (`VERMELHO`)
- **Ciano** (`CIANO`)
- **Branco** (`BRANCO`)

Esses códigos são usados para tornar a interface mais intuitiva ao usuário.

```c
#define VERMELHO "\033[31m"
#define CIANO "\033[36m"
#define BRANCO "\033[37m"
```

1.2 Funções

1.2.1 `void imprimir_menu()`

Exibe um menu com as opções de operações de conjuntos, facilitando a escolha da ação que o usuário deseja realizar. Esta função não recebe parâmetros e não retorna valores.

```c
void imprimir_menu()
{
    printf("0. Sair\n");
    printf("1. Inserir novos conjuntos A e B\n");
    printf("2. A U B\n");
    printf("3. A ∩ B\n");
    printf("4. A - B\n");
    printf("5. B - A\n");
    printf("6. A Δ B\n");
    printf("7. A x B\n");
}
```

1.2.2 `bool elemento_esta_no_conjunto(int elemento, int *conjunto, int len)`

Verifica se um determinado elemento está presente no conjunto. Esta função recebe um elemento a ser buscado, um ponteiro para o conjunto (array de inteiros) e o tamanho do conjunto. Retorna `true` se o elemento for encontrado, e `false` caso contrário.

```
bool elemento_esta_no_conjunto(int elemento, int *conjunto, int len)
{
    int i;
    for (i = 0; i < len; i++)
    {
        if (conjunto[i] == elemento)
        {
            return true;
        }
    }
    return false;
}
```

1.2.3 `int *ler_conjunto(char letra_conjunto, int *len)`

Lê os valores de um conjunto inseridos pelo usuário, evitando valores duplicados. Esta função recebe o nome do conjunto (`A` ou `B`) e um ponteiro para o tamanho do conjunto, que será atualizado conforme a entrada. Retorna um ponteiro para o array dinâmico contendo os valores únicos inseridos.

```
int *ler_conjunto(char letra_conjunto, int *len)
{
    printf("Digite o tamanho do conjunto %c: ", letra_conjunto);
    scanf("%d", len);

    int *conjunto = (int *)malloc(*len * sizeof(int));

    int valor, i;
    bool tentou;
    for (i = 0; i < *len; i++)
    {
        tentou = false;
        do
        {
            if (tentou)
            {
                printf("O número %d já está no conjunto %c.\n", valor, letra_conjunto);
            }
            printf("Digite o valor %d: ", i + 1);
            scanf("%d", &valor);
            tentou = true;
        } while (elemento_esta_no_conjunto(valor, conjunto, *len));

        conjunto[i] = valor;
    }

    return conjunto;
}
```

1.2.4 `void imprimir(int *conjunto, int len)`

Imprime os elementos de um conjunto entre chaves ({}), separando os valores por vírgulas. Utiliza a cor definida pela macro CIANO para destacar o conjunto no terminal.

```
void imprimir(int *conjunto, int len)
{
    int i;

    printf(CIANO "{");
    for (i = 0; i < len; i++)
    {
        if (i < len - 1)
        {
            printf("%d, ", conjunto[i]);
        }
        else
        {
            printf("%d", conjunto[i]);
        }
    }
    printf("}\n" BRANCO);
}
```

1.2.5 `int *uniao(int *conjunto_a, int len_a, int *conjunto_b, int len_b, int *len_res)`

Calcula a união entre dois conjuntos, retornando um novo conjunto contendo todos os elementos dos conjuntos A e B, sem duplicatas. A função recebe os dois conjuntos, seus respectivos tamanhos e retorna o conjunto união, atualizando o tamanho do resultado.

```
int *uniao(int *conjunto_a, int len_a, int *conjunto_b, int len_b, int *len_res)
{
    int i, len_atual = 0, max_len = len_a + len_b;
    int *conj_resposta = (int *)malloc(max_len * sizeof(int));

    for (i = 0; i < len_a; i++)
    {
        conj_resposta[len_atual++] = conjunto_a[i];
    }

    for (i = 0; i < len_b; i++)
    {
        if (!elemento_esta_no_conjunto(conjunto_b[i], conjunto_a, len_a))
        {
            conj_resposta[len_atual++] = conjunto_b[i];
        }
    }

    *len_res = len_atual;

    conj_resposta = (int *)realloc(conj_resposta, (*len_res) * sizeof(int));

    return conj_resposta;
}
```

1.2.6 `int *interseccao(int *conjunto_a, int len_a, int *conjunto_b, int len_b, int *len_res)`

Calcula a interseção entre dois conjuntos, retornando um novo conjunto contendo apenas os elementos que estão presentes em ambos os conjuntos A e B. A função recebe os dois conjuntos, seus respectivos tamanhos e retorna o conjunto interseção, atualizando o tamanho do resultado.

```
int *interseccao(int *conjunto_a, int len_a, int *conjunto_b, int len_b, int *len_res)
{
    int i, len_atual = 0, max_len = min(len_a, len_b);

    int *conj_resposta = (int *)malloc(max_len * sizeof(int));

    for (i = 0; i < len_a; i++)
    {
        if (elemento_esta_no_conjunto(conjunto_a[i], conjunto_b, len_b))
        {
            conj_resposta[len_atual++] = conjunto_a[i];
        }
    }

    *len_res = len_atual;

    conj_resposta = (int *)realloc(conj_resposta, (*len_res) * sizeof(int));

    return conj_resposta;
}
```

1.2.7 `int *diferenca(int *conjunto_a, int len_a, int *conjunto_b, int len_b, int *len_res)`

Calcula a diferença entre dois conjuntos. Retorna um novo conjunto contendo todos os elementos do conjunto A que não estão presentes no conjunto B. A função recebe os dois conjuntos, seus respectivos tamanhos e retorna o conjunto diferença, atualizando o tamanho do resultado.

```
int *diferenca(int *conjunto_a, int len_a, int *conjunto_b, int len_b, int *len_res)
{
    int i, len_atual = 0, max_len = len_a;
    int *conj_resposta = (int *)malloc(max_len * sizeof(int));

    for (i = 0; i < len_a; i++)
    {
        if (!elemento_esta_no_conjunto(conjunto_a[i], conjunto_b, len_b))
        {
            conj_resposta[len_atual++] = conjunto_a[i];
        }
    }

    *len_res = len_atual;

    conj_resposta = (int *)realloc(conj_resposta, (*len_res) * sizeof(int));
```

```
    return conj_resposta;
}
```

1.2.8 `int *diferenca_simetrica(int *conjunto_a, int len_a, int *conjunto_b, int len_b, int *len_res)`

Calcula a diferença simétrica entre dois conjuntos. A diferença simétrica é o conjunto de elementos que estão em A ou em B, mas não em ambos. A função faz uso da união e da interseção para calcular a diferença simétrica. Ela retorna o conjunto resultante e atualiza o tamanho do resultado.

```
int *diferenca_simetrica(int *conjunto_a, int len_a, int *conjunto_b, int len_b, int *len_res)
{
    int len_uniao, len_interseccao;
    int *conj_uniao = uniao(conjunto_a, len_a, conjunto_b, len_b, &len_uniao);
    int *conj_interseccao = interseccao(conjunto_a, len_a, conjunto_b, len_b, &len_interseccao);

    int *conj_resposta = diferenca(conj_uniao, len_uniao, conj_interseccao, len_interseccao, len_res);

    return conj_resposta;
}
```

1.2.9 `typedef struct { int x; int y; } Coordenada`

Define uma estrutura Coordenada que representa um ponto no plano cartesiano, onde x e y são as coordenadas inteiras. Essa estrutura é usada para representar os pares no produto cartesiano.

```
typedef struct
{
    int x;
    int y;
} Coordenada;
```

1.2.10 `void imprimir_produto_cartesiano(Coordenada *conjunto, int len)`

Imprime o resultado do produto cartesiano entre dois conjuntos, exibindo os pares ordenados de coordenadas no formato {x, y}. A função utiliza a cor CIANO para estilizar a saída.

```
void imprimir_produto_cartesiano(Coordenada *conjunto, int len)
{
    int i;
    Coordenada coordenada;

    printf(CIANO "{");
    for (i = 0; i < len; i++)
    {
        coordenada = conjunto[i];
        if (i < len - 1)
        {
            printf("{%d, %d}, ", coordenada.x, coordenada.y);
        }
        else
        {
            printf("{%d, %d}", coordenada.x, coordenada.y);
        }
    }
    printf("}\n" BRANCO);
}
```

1.2.11 `Coordenada *produto_cartesiano(int *conjunto_a, int len_a, int *conjunto_b, int len_b, int *len_res))`

Calcula o produto cartesiano entre dois conjuntos, resultando em um conjunto de pares ordenados (a, b), onde a pertence ao conjunto A e b pertence ao conjunto B. A função aloca dinamicamente um array de estruturas Coordenada e retorna o resultado, além de atualizar o tamanho do conjunto resultante.

```
Coordenada *produto_cartesiano(int *conjunto_a, int len_a, int *conjunto_b, int len_b, int *len_res)
{
    int i, j, len_atual = 0;
    Coordenada *plano_cartesiano = (Coordenada *)malloc(len_a * len_b * sizeof(Coordenada));
    Coordenada coordenada;

    for (i = 0; i < len_a; i++)
    {
        for (j = 0; j < len_b; j++)
        {
            coordenada.x = conjunto_a[i];
```

```
            coordenada.y = conjunto_b[j];

            plano_cartesiano[len_atual++] = coordenada;
        }
    }

    *len_res = len_atual;

    plano_cartesiano = (Coordenada *)realloc(plano_cartesiano, (*len_res) * sizeof(Coordenada));

    return plano_cartesiano;
}
```

1.2.12 `int main()`

Função principal que gerencia a interação com o usuário, incluindo a leitura dos conjuntos, a exibição do menu e a execução das operações escolhidas (união, interseção, diferença, diferença simétrica e produto cartesiano). O código também lida com a alocação e liberação dinâmica de memória.

```
int main()
{
    SetConsoleOutputCP(CP_UTF8);

    int len_a, len_b, len_res;
    int *conj_a = ler_conjunto('A', &len_a);
    int *conj_b = ler_conjunto('B', &len_b);
    int *conj_resposta;
    int opcao;
    Coordenada *plano_cartesiano;

    while (true)
    {
        imprimir_menu();

        printf("Opção: ");
        scanf("%d", &opcao);

        switch (opcao)
        {
        case 0:
            free(conj_a);
            free(conj_b);
            return 0;

        case 1:
            conj_a = ler_conjunto('A', &len_a);
            conj_b = ler_conjunto('B', &len_b);
            break;

        case 2:
            conj_resposta = uniao(conj_a, len_a, conj_b, len_b, &len_res);
            imprimir(conj_resposta, len_res);
            break;

        case 3:
            conj_resposta = interseccao(conj_a, len_a, conj_b, len_b, &len_res);
            imprimir(conj_resposta, len_res);
            break;

        case 4:
            conj_resposta = diferenca(conj_a, len_a, conj_b, len_b, &len_res);
            imprimir(conj_resposta, len_res);
            break;

        case 5:
            conj_resposta = diferenca(conj_b, len_b, conj_a, len_a, &len_res);
            imprimir(conj_resposta, len_res);
            break;

        case 6:
            conj_resposta = diferenca_simetrica(conj_a, len_a, conj_b, len_b, &len_res);
            imprimir(conj_resposta, len_res);
            break;

        case 7:
            plano_cartesiano = produto_cartesiano(conj_a, len_a, conj_b, len_b, &len_res);
            imprimir_produto_cartesiano(plano_cartesiano, len_res);
            break;

        default:
            printf(VERMELHO "Opção inválida\n" BRANCO);
            break;
        }
    }

    free(conj_a);
    free(conj_b);
```

```
    return 0;
}
```